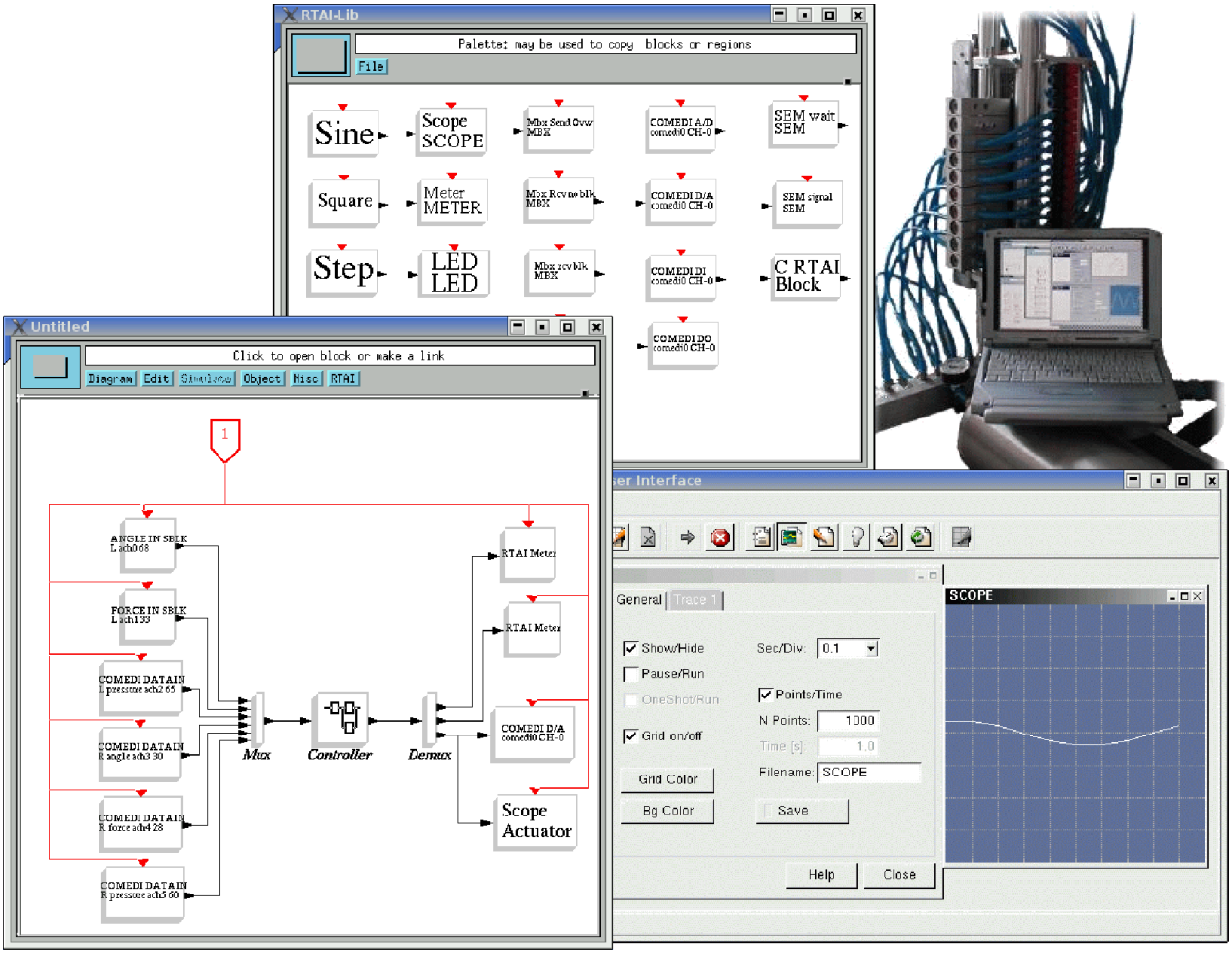
RTAI-Lab tutorial: Scicoslab, Comedi, and real-time control

Roberto Bucher [1] Simone Mannori Thomas Netter [2]

May 24, 2010

Summary

RTAI-Lab is a tool chain for real-time software and control system development. This tutorial shows how to install the various components: the RTAI real-time Linux kernel, the Comedi interface for control and measurement hardware, the Scicoslab GUI-based CACSD modeling software and associated RTAI-Lab blocks, and the xrtailab interactive oscilloscope. RTAI-Lab's Scicos blocks are detailed and examples show how to develop elementary block diagrams, automatically generate real-time executables, and add custom elements.



[1] Main RTAI-Lab developer, person to contact for technical questions: `roberto.bucher at supsi.ch`, see page 46

Contents

1 Introduction **4**
- 1.1 RTAI-Lab tool chain . . . 4
- 1.2 Commercial software . . . 4

2 Installation **5**
- 2.1 Requirements . . . 5
  - 2.1.1 Hardware requirements . . . 5
  - 2.1.2 Software requirements . . . 6
- 2.2 Mesa library . . . 7
- 2.3 EFLTK library . . . 7
- 2.4 Linux kernel and RTAI patch . . . 7
- 2.5 Comedilib . . . 8
- 2.6 RTAI (1st pass) . . . 8
- 2.7 RTAI tests . . . 9
- 2.8 Comedi . . . 9
- 2.9 RTAI (2nd pass) . . . 10
- 2.10 ScicosLab . . . 11
- 2.11 RTAI-Lab add-ons to Scicoslab-4.4 . . . 11
- 2.12 User configuration for scicoslab-4.4 . . . 11
- 2.13 Load the modules . . . 11

3 Development with RTAI-Lab **13**
- 3.1 Boot Linux-RTAI . . . 13
- 3.2 Start Scicos . . . 13
- 3.3 RTAI-Lib palette . . . 14
- 3.4 Real-time sinewave: step by step . . . 16
  - 3.4.1 Create block diagram . . . 16
  - 3.4.2 Compile . . . 17
  - 3.4.3 Execute . . . 19
  - 3.4.4 Change parameters . . . 20
  - 3.4.5 Stop executable . . . 20

4 Examples **22**
- 4.1 FIFOs . . . 22
- 4.2 Semaphores . . . 23
- 4.3 From continuous- to discrete-time . . . 26
  - 4.3.1 Discretize the system . . . 26
  - 4.3.2 Block diagram and simulation . . . 26
  - 4.3.3 Real-time model and simulation . . . 28
- 4.4 DC Motor control . . . 29
  - 4.4.1 Motor position model . . . 29
  - 4.4.2 Digital model . . . 29
  - 4.4.3 Digital control . . . 30
  - 4.4.4 RTAI-Lab controller . . . 31

5 Custom blocks **34**
- 5.1 Overview of Scicos blocks . . . 34
- 5.2 Hardware example: parallel port LED blinker . . . 35
  - 5.2.1 Parallel port + LED circuit . . . 35
  - 5.2.2 Create the environment for the new blocks . . . 35
  - 5.2.3 Generate the interface files . . . 36
  - 5.2.4 Compile the interface files . . . 36
  - 5.2.5 Generate the implementation files . . . 37
  - 5.2.6 Compile the implementation files . . . 37
  - 5.2.7 Integrate the new files in the Scicoslab environment . . . 38

5.2.8 The template Makefile for the new files . . . 38
5.2.9 Create a scicos block palette for the new files . . . 39
5.3 Scicos C block data structure . . . 39

A Notes on Linux configuration and installation **41**
A.1 Kernel configuration . . . 41
A.2 Configuring the GRUB boot manager . . . 42
A.3 *udev* and RTAI . . . 43

B Reference: standard parallel port **43**

References **45**

C About this document / Bug reports **46**
C.1 About the authors . . . 46
C.2 Bug reports, suggestions . . . 46
C.3 Software licenses . . . 46
C.4 Acknowledgements . . . 46

D Useful files and links **47**

1 Introduction

This tutorial shows how to install, use, and customize, a free[1], open-source, Computer Aided Control System Design (CACSD) and real-time control tool-chain called RTAI-Lab.
Apart from being a tool to develop real-time control systems, RTAI-Lab also lets you integrate real-time controllers and simulators generated by Mathworks' Matlab®/Simulink®/Real-Time Workshop® and/or the open source Scicoslab/CodeGen CACSD software suite.
Furthermore, RTAI-Lab lets you:

- Develop and execute real-time software in a local/remote/distributed way,
- Monitor a controller's local/remote/distributed execution,
- Change a controller's parameters on the fly.

1.1 RTAI-Lab tool chain

The RTAI-Lab tool-chain is based on:

- **Scicoslab**. Scicoslab is an open source CACSD software for numerical computation. Scicoslab includes Scicos, a block diagram editor that can be used to create simulations and automatically generate and compile code. See www.scicoslab.org and www.scicos.org
- **Comedi.** Comedi provides the drivers, library functions, and an API to interact with signal acquisition hardware. Hundreds of devices are supported. See www.comedi.org
- **RTAI**. The Real-Time Application Interface (RTAI) is distributed as a package with a patch to apply to the Linux kernel. RTAI inserts a sub-kernel where prioritized, hard real-time tasks can run. FIFOs and shared memory can be used to transfer data between real-time and user space processes. See www.rtai.org
- **RTAI-Lib**. RTAI-Lib is a palette of Scicos blocks that let you design block diagrams with sensors and actuators. It provides an interface to RTAI and signal acquisition hardware. Block diagrams that use RTAI-Lib can be compiled into RTAI execuatble software. It is included in the RTAI package.
- **xrtailab**. xrtailab is an oscilloscope-like software that can connect to your real-time executables. It lets you visually monitor signals and real-time events using gauge, scope, and LED mock-ups. xrtailab also lets you adjust parameters of the real-time executable while it runs. It is also part of RTAI.

1.2 Commercial software

All software used for RTAI-Lab is free and Open Source. The RTAI-Lab tool chain represents an alternative to the commercial software listed below:

- Scicoslab → Mathworks'
  Matlab®/Simulink®
- Comedi → drivers supplied by signal acquisition hardware vendors, but most vendors don't supply Linux drivers
- RTAI → LynxOS® from Lynuxworks®, MontaVista Linux®, QNX®, VxWorks®, etc.
- RTAI-Lib → Mathworks' Real-Time Workshop®
- xrtailab → LabView®

[1] No purchase cost. See licenses in Appendix C.3, page 46

Note that you can still use commercial software and use RTAI/RTAI-Lab as a target (see RTAI-Target-HOWTO and RTAI-UbuntuGutsy-Matlab in Appendix D). If you think you have sufficient funds to purchase commercial software, you will then buy a better GUI, better documentation, support that you can pay for, and the possibility to consult with the vendor in case of problems (especially if you work for a large company). On the other hand, upgrades with bug fixes are expensive, should be budgeted, and included in your real-time software's multi-year maintenance and extension plan. These costs represent a financial burden even for companies with a few hundred employees and only a few licenses. The advantage of using exclusively Open Source software in the RTAI-Lab tool chain is that you or a subcontractor can customize it. You probably already know that. That's why softwares that constitute the RTAI-Lab tool chain are developed.

2 Installation

The full installation requires several software packages to be downloaded from their development servers and compiled from source. Since RTAI, RTAI-Lab, and Comedi are undergoing strong development we recommend that you do not rely on precompiled packages provided by Linux distributions.
The following sections provide detailed instructions to install (order is important and must be followed). If you are installing on an Ubuntu-based distribution you may also refer to www.rtai.org/RTAILAB/RTAI-UbuntuGutsy-Matlab.txt

- The Mesa 3D graphical library. http://www.mesa3d.org
- The EFLTK graphic widgets library. equinox-project.org
- A new Linux kernel that will be patched with RTAI code
- The Comedilib data acquisition device interface library
- The RTAI real-time modules
- The Comedi data acquisition device modules
- The Scicoslab Computer Aided Control System Design (CACSD) software

Note: The next sections list commands where software minor versions are indicated with "x.x". Make sure to replace the "x" with version numbers that correspond to your downloads.

2.1 Requirements

2.1.1 Hardware requirements

Most PCs can run a Linux kernel patched with RTAI. Some hardware configurations deserve a specific comment.
Comedi support. Before starting the installation, first check that your data acquisition card or external hardware is supported by Comedi. See (this page may be out of date) http://www.comedi.org/hardware.html. If you don't see your hardware listed then download Comedi anyway (see section 2.5) and look/`grep` into `/usr/local/src/comedi/comedi/drivers`
PCMCIA. Our PCMCIA card was not detected with some Linux kernels 2.6.13.xx, 2.6.14.xx, 2.6.19 (see note in Sec. 2.4).
Laptops & PCI sharing. We have experienced incompatibilities with some Toshiba Satellite laptops. If you use a data acquisition card that plugs into the PCMCIA/CardBus slot of a Toshiba Satellite laptop you might see warning messages about PCI devices sharing the same IRQ when loading Comedi modules. RTAI+Comedi might not work, but give it a try anyway. If you know of other laptops that have this problem, or know how to solve the problem, please notify us.
A quick way to test whether your computer will easily work with RTAI is to download an ISO image from issaris.homelinux.org/ takis/projects/rtai/livecd and conduct some latency tests. Note that failing to boot the CD or run the tests does not necessarily mean your computer cannot run RTAI.

2.1.2 Software requirements

We assume that your Linux system is based on one of the available distributions such as Debian, Fedora, Mandriva, Red Hat, Slackware, Suse, Ubuntu, etc., and that you have suitable software development libraries installed to compile a Linux kernel, as well as graphical development libraries.
Your working environment and list of packages should enable you to compile and install software as `root`. Ensure that your distribution is up to date with the latest versions of development software such as `automake > 1.7`, `autoconf`, `libtool`, `bison`, and `doxygen`.

2.2 Mesa library

New Linux distributions (in particular Ubuntu) already contain usable Mesa libraries.

2.3 EFLTK library

EFLTK is needed by `xrtailab`. EFLTK is part of the EDE project ede.sourceforge.net. You will need packages gettext, flex, and svn.

1. `cd /usr/local/src`
   `svn co https://ede.svn.sourceforge.net/svnroot/ede/trunk/efltk`
   `svn co https://ede.svn.sourceforge.net/svnroot/ede/trunk/ede`

2. `cd efltk`
   `autoconf`
   `./configure --disable-mysql --disable-unixODBC`
   `./emake`
   `./emake install`

3. Edit `/etc/ld.so.conf` and add a line with the path `/usr/local/lib` on a line.

4. Execute `/sbin/ldconfig` to update the library database

2.4 Linux kernel and RTAI patch

- `cd /usr/src`
- Download RTAI (at least version 3.8) of either:
  - Stable (numbered): `http://www.rtai.org` (e.g. `rtai-3.8.tar.bz2`)
    `tar xjvf rtai-3.8.tar.bz2`
    `ln -s /usr/src/rtai-3.8 rtai`
  - Experimental (hot stuff, beware): `cvs -d:pserver:anonymous@cvs.gna.org:/cvs/rtai co vulcano`
    (see also `https://gna.org/cvs/?group=rtai`)
- Check kernel version numbers of available RTAI patches:
  `ls /usr/src/rtai/base/arch/x86/patches/`
  and look for kernel version numbers in: `hal-linux-<kernel-version>.patch`
- **Kernel 2.6.xx.** The versions supported by this RTAI release can be found in the RTAI-3.8/base/arch/x86/patches directory

  1. Download a ”vanilla” linux-2.6.xx.xx.tar.bz2 from `http://www.kernel.org`
  2. Unpack the kernel: `tar xjvf linux-2.6.xx.xx.tar.bz2`
     `ln -s /usr/src/linux-2.6.xx.xx linux`
  3. Patch the kernel:
     `cd /usr/src/linux`
     `patch -p1 < <rtaidir>/base/arch/x86/patches/<kernel-version>.patch`
  4. `make xconfig` or `make menuconfig`
  5. Configure the kernel. See details in Appendix A
  6. `make`
  7. `make modules_install`
  8. `make install`

- **Update your boot loader** (either `lilo` or `grub`):

  - Lilo: edit `/etc/lilo.conf`, and execute `lilo`
  - Grub: edit `/boot/grub/menu.lst` (Debian) or `/etc/grub.conf` (Fedora, see Appendix B)
- Add module names to `/etc/modules` (Debian), e.g. `pcmcia` if you use a laptop with a signal acquisition card and execute `update-modules` (Debian)
- **Reboot computer into newly compiled kernel**

2.5 Comedilib

1. `cd /usr/local/src`
2. ```
   cvs -d :pserver:anonymous@cvs.comedi.org:/cvs/comedi login
   cvs -d :pserver:anonymous@cvs.comedi.org:/cvs/comedi co comedi
   cvs -d :pserver:anonymous@cvs.comedi.org:/cvs/comedi co comedilib
   ```
3. `cd comedilib`
4. Read software installation requirements in `README.CVS` and verify that your packages (automake etc.) are up to date with `automake --version`
5. `sh autogen.sh`
   Note: you may ignore warnings and reminders
6. `./configure --sysconfdir=/etc`
   Note: pay attention to warnings and possibly remedy to these by installing extra software/packages
7. `make` and `make install`
   Note: installation places `comedi.h` and `comedilib.h` in `/usr/local/include`. You will later have to overwrite these header files with those from Comedi (see Sec. 2.8).
8. `make dev`
   This step creates the `/dev/comedi`[0-3] device inodes.
9. Optional calibration tool: `comedi_calibrate`
   Install packages from your favourite Linux distribution: `libboost-program-options-dev` and `libgsl0-dev`
   See also RTAI manual Sec. 4.7-8.

2.6 RTAI (1st pass)

This is usually straightforward. For more details download a draft of the RTAI User Manual Racciu and Mantegazza [2006]. You will need packages libxmu-dev and libxi-dev.

1. `cd /usr/src/rtai`)
2. `make xconfig` or `make menuconfig`
3. Menu General: verify default directories:
   - Installation directory `/usr/realtime`
   - Kernel source directory `/usr/src/linux`
4. Menu General: optionally select RTAI Documentation (HTML, PDF,...)
5. Menu Machine (x86): adjust Number of CPUs (default = 2)
6. Exit xconfig/menuconfig and save configuration
7. `make` and `make install`
8. **IMPORTANT:** Add `/usr/realtime/bin` to the PATH variable in /etc/profile or your home directory's `.bashrc`.
   ```

2.7 RTAI tests

Load RTAI-related modules (Section 2.13, `insmod` modules `rtai_hal` to `rtai_fifos`). You should measure your system's latency to real-time interrupts:
`cd /usr/realtime/testsuite/kern/latency`
`./run`
This launches a periodic task. The default period of 100000 ns is defined by `DEFAULT_PERIOD` in `/usr/src/rtai/testsuite/kern/latency/latency-module.c` and can be redefined upon module insertion (see below). Watch *lat min* (ns) and *overruns*. Load your computer's CPU, e.g. use multimedia applications. If *overruns* becomes greater than zero then you might try to lenghten the period:

- Type `ctrl-c` to stop execution
- `insmod` modules of Section 2.13 from `rtai_hal.ko` to `rtai_fifos.ko`
- `insmod /usr/realtime/modules/rtai_lxrt.ko`
  `insmod latency_rt.ko period=1000000`
  `./display`
  Overruns should either remain at zero or increase much more slowly. If they are still increasing then stop execution and reload `latency_rt.ko` with a longer period.

Stop execution: `ctrl-c`
`rmmod latency_rt`
You may compare your latency results with those published at issaris.org/˜ takis/projects/rtai → Benchmarks.

2.8 Comedi

In case of difficulties, please refer to the INSTALL file in the `comedi` directory and to the documentation at www.comedi.org.
IMPORTANT: Make any edits that may be requested in `/usr/src/rtai/base/arch/i386/patches/README`, especially if they apply to the Linux kernel's `.config` because it is read by Comedi's `configure` script. See also earlier note in Sec. 2.4 if you use a PCMCIA card.

1. `cd /usr/local/src/comedi`
2. `sh autogen.sh`
3. `./configure` or possibly:
   `./configure --with-linuxdir=/usr/src/linux --with-rtaidir=/usr/realtime`
   or: `./configure --with-linuxdir=/usr/src/linux --with-rtaidir=/usr/realtime --enable-kbuild`
   with PCMCIA acquisition card: `./configure --with-linuxdir=/usr/src/linux --with-rtaidir=/usr/realtime --enable-pcmcia`
   Note: Towards the end of the output pay attention that either CONFIG_IPIPE, CONFIG_RTHAL, or CONFIG_ADEOS is set to "yes". This reflects what is set in `/usr/src/linux/.config`
4. `make`
5. `make install` (installs the comedi kernel modules)
6. `make dev`
7. `cp include/linux/comedi.h include/linux/comedilib.h /usr/include/`
   `cp include/linux/comedi.h include/linux/comedilib.h /usr/local/include/`
8. `ln -s /usr/include/comedi.h /usr/include/linux/comedi.h`
   `ln -s /usr/include/comedilib.h /usr/include/linux/comedilib.h`
9. Load RTAI and Comedi modules (Section 2.13)
10. Configure Comedi to work with you data acquisition hardware (section 2 of Comedi documentation www.comedi.org/doc). The Comedi manual and `man comedi_config` show how to associate a particular driver and hardware device to one of the `/dev/comedi` device files. For example, to configure a National Instruments 6024E PCMCIA card, type:
    `comedi_config -v /dev/comedi0 ni_mio_cs`
    and the output is:
    `configuring driver=ni_mio_cs 0,0,0,

2.9 RTAI (2nd pass)

1. `cd /usr/src/rtai`
2. `make xconfig` or `make menuconfig`
3. Menu Add-ons:
   - Select COMEDI support over LXRT
   - Specify COMEDI installation directory (`/usr/local` or `/usr`). The directory should contain `lib/libcomedi.a`, `include/comedi.h` and `include/comedilib.h`

4. Menu RTAI Lab:
   - Select RTAI Lab
   - Adjust EFLTK installation directory (default is `/usr/local`)
5. Exit xconfig/menuconfig and save configuration
6. `make` and `make install`

2.10 ScicosLab

The binary installation file for different Linux distributions can be downloaded at the project homepage www.scicoslab.org.
The Scicoslab add-ons for thenew Scicoslab-4.4 version are provided in vulcano-cvs.

2.11 RTAI-Lab add-ons to Scicoslab-4.4

1. `cd /usr/src/vulcano/rtai-lab/scicoslab/macros`
2. Adjust Scicoslab version and installation directory in `Makefile`, e.g.:
   `SCILAB_VERSION = 4.4b7`
3. `make install`

2.12 User configuration for scicoslab-4.4

Every RTAI-Lab user must launch the previous make with

- make user

or edit the
`${HOME}/.Scilab/4.x/.scilab` with:

```
exec("SCI/contrib/RTAI/loader.sce");
```

These lines add:

- Menu RTAI to the Scicos diagram editor with items →CodeGen and →Set Target
- Palette RTAI-Lib to menu Edit→Palettes

To add Scicoslab to the executable path you must either:

```
cd /usr/local/bin
ln -s /usr/lib/scicoslab-gtk-4.x/bin/scilab scilab
```

or add `/usr/lib/scicoslab-4.4/bin` to the PATH variable in your `.bash_profile and/or .bashrc` or relevant shell start-up file.

2.13 Load the modules

To compile and execute real-time RTAI-Lab programs you must boot the Linux-RTAI kernel. If all goes well your acquisition hardware will be detected (try `cardctl info` for kernels less that 2.6.13 or `pccardctl info` otherwise). You then load RTAI- and hardware-related modules. Note that which modules to load depends on your RTAI kernel configuration.
We recommend to initially load the modules manually, and possibly later configure your system to load these modules at boot time using system files or commands (depending on your Linux distribution and hardware) such as `/etc/modules, update-modules, /etc/modutils/..., /etc/hotplug/ or /etc/udev` and/or a manually launched script.
To manually load the modules, you will have to be root or use `sudo`. You can check the progress by issuing `dmesg` commands or `tail -f /var/log/messages`. For example, on a laptop with a National Instruments DAQCard-6024E PCMCIA card (see hardware-specific comments indicated by # signs below), here's how to load the modules:

```
insmod /usr/realtime/modules/rtai_hal.ko
insmod /usr/realtime/modules/rtai_up.ko         # or rtai_lxrt.ko
insmod /usr/realtime/modules/rtai_fifos.ko
insmod /usr/realtime/modules/rtai_sem.ko
insmod /usr/realtime/modules/rtai_mbx.ko
insmod /usr/realtime/modules/rtai_msg.ko
insmod /usr/realtime/modules/rtai_netrpc.ko  ThisNode="XXX.XXX.XXX.XXX"
```

Note that you may have to load more modules in `/usr/realtime/modules/` depending on what blocks you use in your application. It may be useful to write a shell script to load these modules after you boot RTAI.

3 Development with RTAI-Lab

Real-time software development with the RTAI-Lab tool chain is done with the Scicoslab block diagram editor and associated palettes of blocks, including the RTAI-Lib palette.. Some testing of the real-time executable can be done with `xrtailab`. New Scicos blocks may be programmed using the Scicoslab language, and their executable component programmed in C (see section 5).

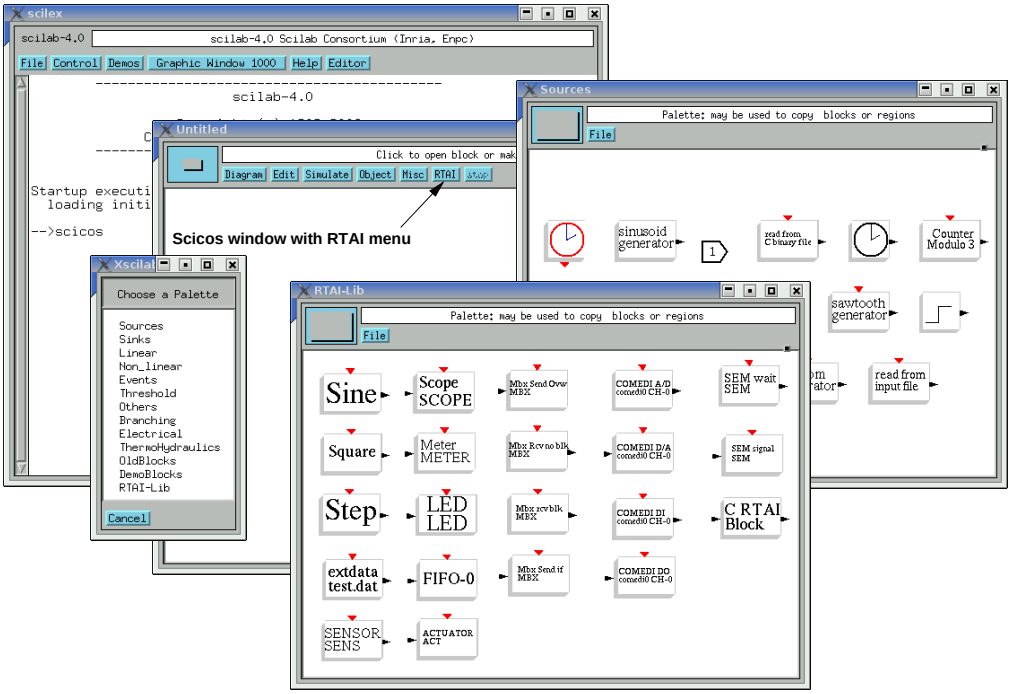


Figure 1: Scicoslab, Scicos, and 2 palettes (Sources and RTAI-Lib). Note the addition of the RTAI drop-down menu button to the standard Scicos window. The most important block in the Sources palette is the red clock.

3.1 Boot Linux-RTAI

Boot Linux-RTAI. Load associated modules (section 2.13).

3.2 Start Scicos

To launch Scicoslab, type at a shell's command prompt: `scicoslab`
At Scicoslab's prompt type: `scicos`
This opens the Scicos window in which you will draw block diagrams. For a tutorial on Scicos check http://www.scicos.org/TUTORIAL/tutorial.html
RTAI-Lab modifies 2 aspects of Scicos (Fig.1):

- There is an RTAI menu towards the right of the Scicos window
- There is an RTAI-Lib palette

3.3 RTAI-Lib palette

RTAI-Lab provides the Scicos RTAI-Lib palette (Fig. 2). To display the palette from Scicos click on menu Edit → Palettes, a pop-up opens, then click on RTAI-Lib which should be listed at the bottom of the pop-up.

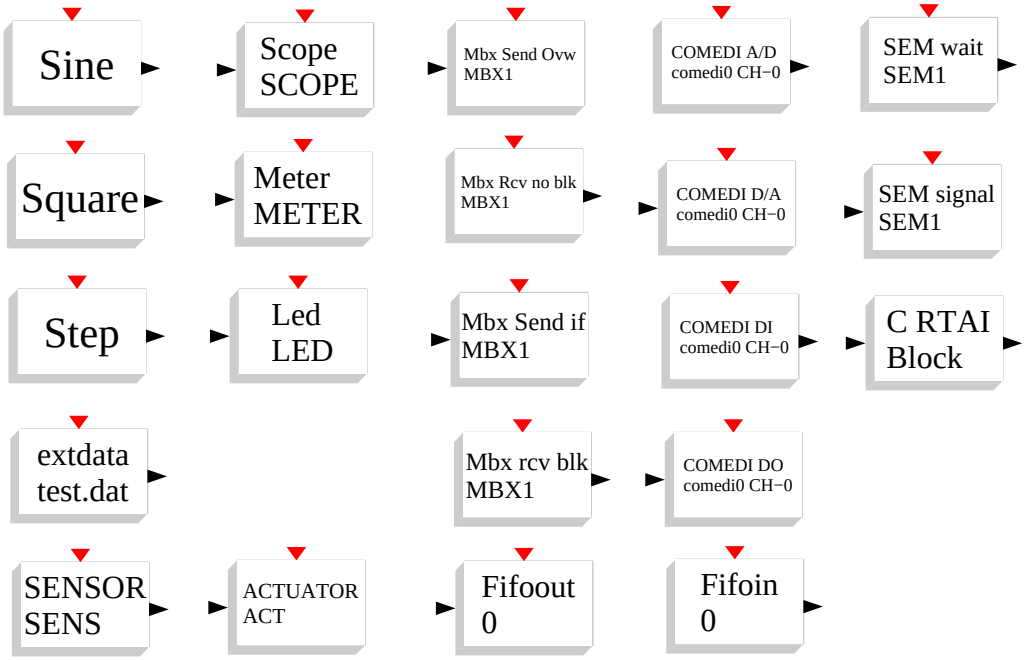


Figure 2: The RTAI-Lib palette featuring (from left to right columns) inputs, outputs, mailboxes, hardware interfaces, and semaphores.

The blocks are (more or less) arranged in thematic columns:

1. **Inputs:** function generators (sine, square, step), read data file, programmable generic sensor
2. **Outputs:** displays (oscilloscope, meter, LED), programmable generic actuator
3. **Mailboxes:** send message and overwrite, receive without blocking task, receive unconditionally, send message if task timing permits, and FIFO boxes
4. **Comedi hardware drivers:** A/D, D/A, digital input, digital output
5. **Semaphores and more:** wait, signal, programmable generic C block

If you have correctly installed RTAI-Lab you can find the source code for the blocks in `/usr/local/scilab-4.x.x/macros/RTAI`. In any case the original blocks provided with the RTAI distribution are in `/usr/src/rtai/rtai-lab/scilab/macros/RTAI`. Most of these blocks call up RTAI-Lab functions written in C and compiled into the library `/usr/realtime/lib/libsciblk.a`. The source code of the C functions is in `/usr/src/rtai/rtai-lab/scilab/devices`. If you look at the source code of the RTAI-Lab functions, you will see how RTAI API functions are called. These RTAI API functions are documented in HTML and other formats in `/usr/src/rtai/doc/generated` if you specified that documentation be generated during RTAI installation (section 2.6).
A detailed review of RTAI-Lib blocks follows. Default values are indicated in brackets.

1. **Inputs:**
   - **Sine.** Real-time sinewave generator. The Sine block lets you adjust amplitude [1], frequency [1] (Hz), phase [0], bias [0], and delay [0]. Furthermore, you can adjust these parameters from xrtailab or an external program while the real-time executable is running.

- **Square.** Real-time square waveform. The Square block lets you adjust amplitude [1], period [4] (s), pulse width [2] (s), bias [0], and delay [0] (s). You can also adjust parameters from xrtailab or other programs during real-time execution.
- **Step.** Step function. The Step block lets you adjust amplitude [1] and delay [0] (s).
- **extdata.** Load data from a file. The file [test.dat] must contain a single column with ASCII values at each sampling time [1000 points].
- **SENSOR.** Generic sensor input. You can adjust the number of outputs [1] and an identifier [SENS]. Following that another dialog box opens where you can enter C code. This code is compiled and linked after clicking on `OK`. Although the dialog box has basic editing functions similar to Emacs, we recommend that you edit your code in a separate editor and then copy-paste it into the dialox box.

2. **Outputs:**
   Scope, Meter, and LED are output blocks that send data to `xrtailab` displays.

   - **Scope.** `xrtailab` multichannel oscilloscope. You can adjust the number of inputs [1] and the scope's name [SCOPE].
   - **Meter.** `xrtailab` single channel meter. You can adjust the meter's name [METER].
   - **LED.** `xrtailab` multichannel multi-LED block. Each LED switches ON if the input is positive. You can adjust the number of inputs/LEDs [1] and the LED block's name [LED].
   - **ACTUATOR.** Generic actuator output. This block is similar to the SENSOR block. You can adjust the number of inputs [1] and an identifier [ACT].

3. **Mailboxes:**
   Mailboxes let real-time tasks exchange arbitrarily-seized messages. Their advantage is that there is no need to size mailbox buffers to cater for unusually large messages. Message passing may be interrupted by real-time constraints. See /usr/src/rtai/doc/generated/html/api/group__mbx.html and [Sarolahti, 2001]
   Local task blocking may occur when exchanging a packet between a local host and a mailbox on a remote host. Furthermore, packet transmission to a mailbox on a remote host is currently limited by the UDP packet size ($\approx$ 1500 bytes).

   - **Mbx Send Ovw.** Sends a message to a mailbox, possibly overwriting what is already in the mailbox. You can adjust the number of message input ports [1], mailbox name [MBX], and mailbox IP address [127.0.0.1] (localhost). See also RTAI API documentation for function `rt_mbx_ovrwr_send`.
   - **Mbx Rcv no blk.** Receives a message only if the whole message can be passed without blocking the calling task. However, if the mailbox is on a remote host, the calling task may be blocked. You can adjust the number of message output ports [1], mailbox name [MBX], and mailbox IP address [127.0.0.1] (localhost). See also RTAI API documentation for function `rt_mbx_receive_if`.
   - **Mbx Rcv blk.** Receives a message. The calling task will be blocked until all bytes of the message arrive or an error occurs. You can adjust the number of message output ports [1], mailbox name [MBX], and mailbox IP address [127.0.0.1] (localhost). See also RTAI API documentation for function `rt_mbx_receive`.
   - **Mbx Send if.** Sends a message only if the whole message can be passed without blocking the calling task. However, if the mailbox is on a remote host, the calling task may be blocked. You can adjust the number of message input ports [1], mailbox name [MBX], and mailbox IP address [127.0.0.1] (localhost). See also RTAI API documentation for function `rt_mbx_send_if`.
   - **FIFOout and FIFOin.** Multichannel FIFO. You can adjust the number of inputs [1], the FIFO's number [0], and the FIFO's Dimension [50000] (in bytes).

4. **Comedi hardware drivers:**

   - **Comedi A/D.** Comedi supported Analog Input. You can select your acquisition card's analog input channel [0], the device if you have several acquisition cards [comedi0], the range number as specified in your acquisition hardware's manual [0] (warning: the value to enter is not expressed in volts), and the type of voltage reference [0], e.g. nonreferenced single ended, referenced single ended, and differential. See also Comedi API `comedi_data_read`.
   - **Comedi D/A.** Comedi supported Analog Output. The parameters are similar to Comedi A/D. See also Comedi API `comedi_data_write`.
   - **Comedi DI.** Comedi supported Digital Input. You can select the channel [0] and device [comedi0]. See also Comedi API `comedi_dio_read`.
   - **Comedi DO.** Comedi supported Digital Output. You can select the channel [0], device [comedi0], and threshold [1]. If the data input to the block is greater than the threshold value, then a single bit will be output by your hardware. See also Comedi API `comedi_dio_write`.

5. **Semaphores and more:**
   Semaphores can be used to synchronize real-time tasks. Similarly to mailboxes, caution is needed with respect to blocking when exchanging a semaphore between a local and a remote host.

   - **SEM wait.** Decrement a semaphore's value and wait for a signal event. The caller task is blocked and queued up as long as the semaphore's value is negative. You can adjust the semaphore's name [SEM] and IP address [127.0.0.1] (localhost). See also RTAI API documentation for function `rt_sem_wait`.
   - **SEM signal.** Increment a semaphore's value. If the resulting value is not positive then the first task in the semaphore's waiting queue is allowed to run. You can adjust the semaphore's name [SEM] and IP address [127.0.0.1] (localhost). See also RTAI API documentation for function `rt_sem_signal`.
   - **C RTAI.** This is a generic block that lets you edit C code. It is based on Scicoslab's C-Block2 found in the "Others" palette. When you click on the C RTAI Block a dialog window opens where you can edit block parameters such as the name of the main function that it will contain, the number of inputs and outputs for data and events, and various parameters (see section 5.3 and also [Campbell, Chancelier, and Nikoukhah, 2006], section 9.5). After clicking on OK a second window opens with a code skeleton to edit. It is best to copy-paste code from your favorite editor into this window. When you click on OK the code is compiled and linked. If your code is relatively large we recommend that you program your own block instead of using C RTAI Block (see section 5).

3.4 Real-time sinewave: step by step

In this example you use Scicoslab to draw a block diagram that generates a sinewave. You automatically generate and compile a real-time executable. You test the program and visualize the sinewave with `xrtailab`.

3.4.1 Create block diagram

Let's create the preliminary Sinewave diagram shown in figure 3.
Select palettes. Start Scicos. Open the Sources palette: click on menu Edit → Palettes and select Sources at the top of the pop-up window. Open the RTAI-Lib palette in a similar way (figure 1).
Keyboard shortcuts. Note that by default, Scicos has a few customizable keyboard shortcuts, list them with menu Misc → Shortcuts.
Select blocks. From the Sources palette, left-click on the red clock. Move your mouse to the main Scicos window. You should see the block's contour being dragged as your mouse cursor enters the main Scicos window. Left-click again to place the clock in the main Scicos window, somewhere towards the upper center of the window. You may also right-click to cancel block placement during the drag.
From the RTAI-Lib palette, left-click on the Sine block and place it towards the lower left of the main Scicos window. Place also a Scope block towards the lower right.

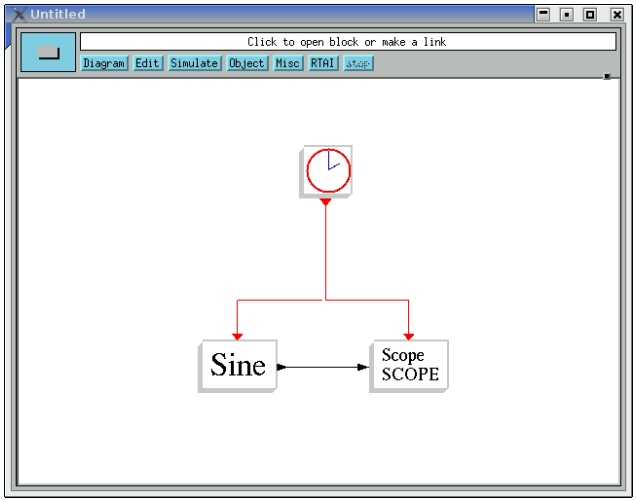


Figure 3: Preliminary RTAI-Lib block diagram with real-time sinewave generator and xrtailab scope blocks.

Align blocks. Scicos does not currently feature a block placement grid so blocks are not perfectly aligned to make a nice diagram. Two methods can help block alignment:

- Use Scicos's Align function: menu Edit → Align. Click on the Sine block's *data output port* (the black triangle to the right of the block) and then on the Scope's *data input port*. One of the blocks will then be shifted so that both blocks' ports are aligned. A faster way to align blocks using keyboard shortcuts is to place the mouse over a block's port (input or output), then use keyboard shortcut "a", then click on the port of the block to be aligned.
- Adjust mouse acceleration so that motion increments are more discretized: type `xset m 4 1` in a terminal/shell window. The drawback is that your mouse might move too fast for comfort.

Link blocks. Menu Edit → Link. Click on Sine's output port. Click on Scope's input port. A link is drawn from Sine to Scope. Click on the clock's *event output port* (the red triangle below the clock). Then draw an S-shaped link to the Sine block by clicking below the clock's port, once more above the Sine's event port, and finally directly onto the Sine's event port. A red link is drawn from the clock to the Sine. Draw another event link by clicking onto the red line, then click above the Scope, and click again onto the Scope's event port.
Set block parameters. Click on the clock. You may adjust period and initialization times (values are in seconds). Click on Sine, adjust parameters accordingly, then click on OK. You may also edit the Scope's name and notice that it may actually have more than one data input. In any case, it is necessary that Scicos *evaluate* all blocks. This is done when you click on a block, adjust its parameters, and then click on OK. A general way to have all blocks *evaluated* is to select menu Simulate → Eval. You must do this Eval operation every time you add a block to a diagram and omit inspecting its internal parameters.
Create super block. Menu Diagram → Region to Super Block. You must draw an elastic frame around the Sine and Scope blocks, excluding the Clock (figure 4, left). First click above-left of the Sine block, then click again below-right of the Scope block. The region then becomes a Super Block (Fig. 4, right).

3.4.2 Compile

Set target. You may optionally click on menu RTAI → Set Target and then middle-click on the Super Block to compile. A dialog box opens where you may modify:

- **Target.** This is the Makefile's basename. See description below.
- **Ode function.** This is one of the ordinary differential equation functions available in Scicoslab. The default is `ode4`. Possible values are:
  - **ode1.** Uses Euler's method (RK1)
  - **ode2.** Uses Heun's method (RK2), also known as the Improved Euler method.

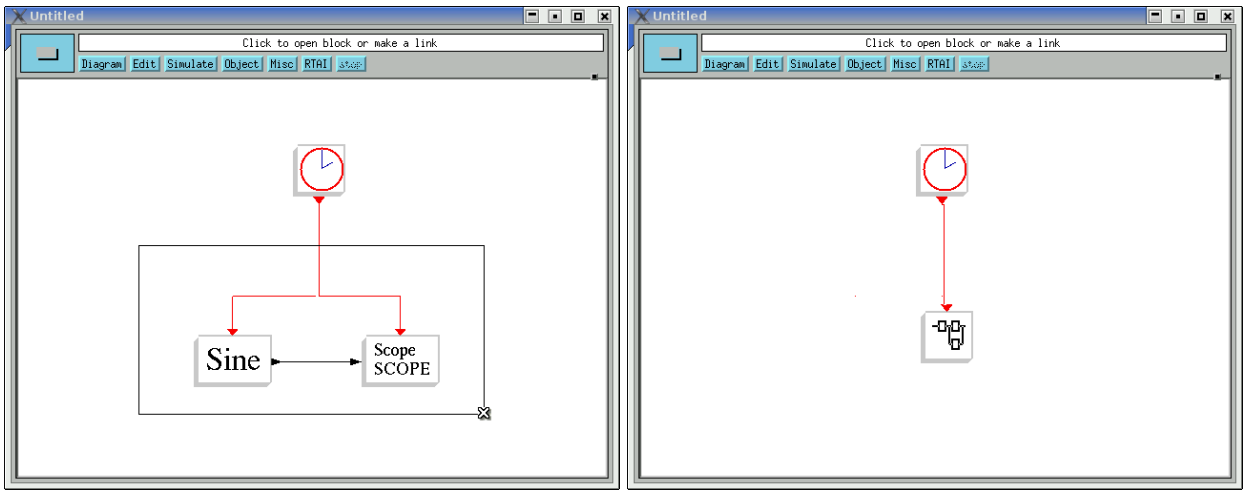
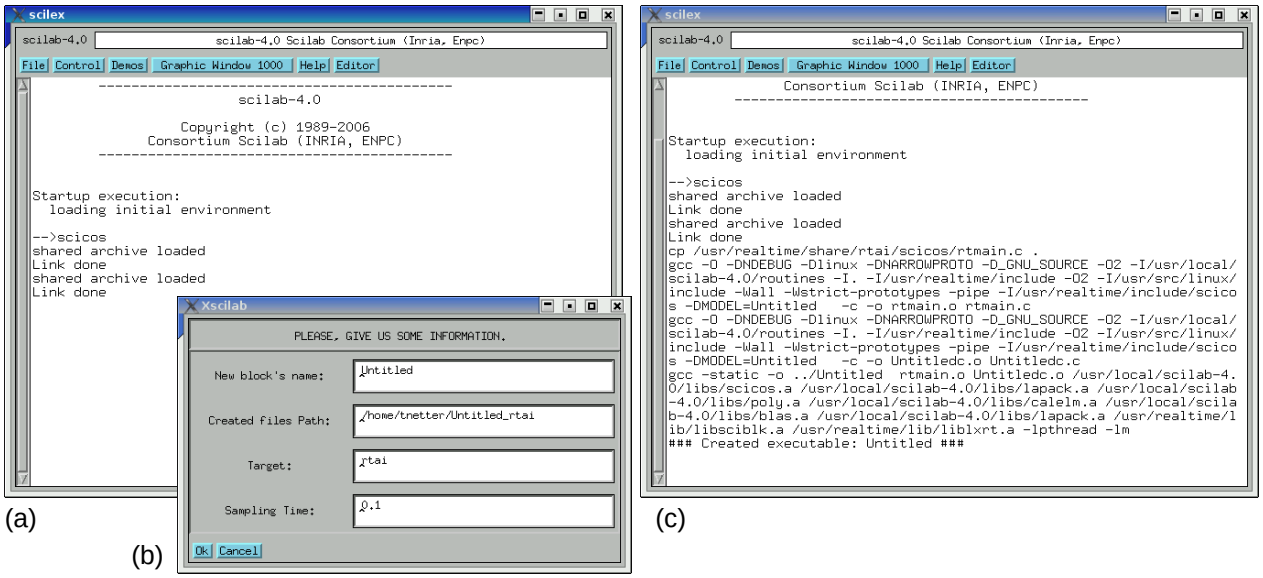


Figure 4: **Left:** preliminary RTAI-Lib block diagram during Region to Super Block framing. All blocks except the clock should be contained in the region. **Right:** resulting Super Block.

  - **ode4.** Uses a 4th order Runge-Kutta formula (RK4).

  The source code for these functions is available in `$SCILAB/macros/RTAI/RTAICodeGen_.sci`. Scicoslab's ODE functions are detailed in section 3.2 of [Campbell et al., 2006] and also in [Sallet, 2004].

- **Step between sampling.** Indicates the number of computational sub-sampling points used for various functions such as ODE functions. The default is 10.



(a) (b) (c)

Figure 5: Block diagram: code generation and compilation. **(a)**: Scicos window after code generation. **(b)**: Dialog box to adjust generated filenames, makefile, and clock period. **(c)**: Text output after compilation.

Compile block diagram. Menu RTAI → RTAI CodeGen. Then click on the Super Block. At this point Scicoslab converts your block diagram into C code. For each block inside the Super Block two lines are output in the Scicos window (Fig. 5a):

```
shared archive loaded
Link done
```

A dialog then opens (Fig. 5b shows the dialog with default values) where you may modify:

- **New block's name.** This will be the name of the final executable. It will be saved in what Scicoslab thinks is your current directory. The default is `Untitled`.

- **Created files Path.** This is the directory where the generated C files are saved along with a Makefile. This is also the directory where compilation occurs.
- **Target.** This is the Makefile's basename. It will be copied to the generated files directory and used for compilation. The default value `rtai` corresponds to the file `$SCILAB/macros/RTAI/RT_templates/rtai.mak`
- **Sampling Time.** This corresponds to the Period value set in the Clock block parameters, i.e. you can adjust the clock here.

Click on OK and the compilation starts. Steps in the compilation can be monitored in the Scicoslab window (Fig. 5c). In case the compilation fails (this can happen for example when you develop your own custom blocks, see section 5), you can still start the compilation manually by typing `make` in the directory where C files were generated. This provides more debugging output that what is printed in the Scicoslab window.
If you kept the default values you should find an executable file called `Untitled` in your current directory.

3.4.3 Execute

Open two terminal shells.
Real-time executable. In the first terminal type: `Untitled -u`
This provides usage instructions as a list of command line options.
To start the real-time executable with verbose output type: `Untitled -v`
xrtailab. Note that `xrtailab` only executes when an RTAI Linux kernel is running. `xrtailab` will generate a segmentation fault if started over a standard Linux kernel.
In the second terminal type:

```
xrtailab -h
xrtailab
```

Select menu File → Connect. . . (keyboard shortcut: alt-c) then Click on OK (Fig. 6).

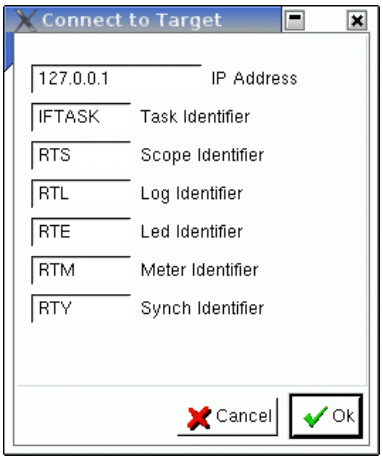


Figure 6: `xrtailab`'s connect dialog box.
IP Address is that of the host where the real-time task is executing. If you are working local you can enter "0" as IP Address.
Identifiers IFTASK, RTS, RTL, RTE, RTM, and RTY are predefined in `rtmain.c`, in the directory that contains the generated source code. If several real-time tasks must run in parallel, a real-time task's identifiers may be set via the command line options (execute the task with option `-u` to list command line options) and by adjusting values in the "Connect to Target" dialog. Note that you may even edit `rtmain.c` to change the real time task's predefined identifiers.

`xrtailab`'s row of buttons (Fig. 7) lets you start and stop the real-time executable and open various displays such as scopes, meters, and LEDs. At the time this tutorial is written, not all buttons are fully functional. Some may find it more convenient to use the View menu and associated keyboard shortcuts.
Open a scope manager window by either clicking on the Scope button or selecting it in the View menu (shortcut: alt-s). Then click on the Show/Hide checkbox to display the scope. You should see a 1 Hz sinewave snaking smoothly in the scope.
Note that the Scope Manager lets you modify display properties such as the number of grid division (Sec/-Div), colors, and filename. You can save data points (default: 1000) to a file (default filename: SCOPE). You may also pause the display. Each trace's properties can also be adjusted via the Trace tab. The trace can be hidden, its offset shifted, and its color changed. You may also adjust the number of units per vertical division.

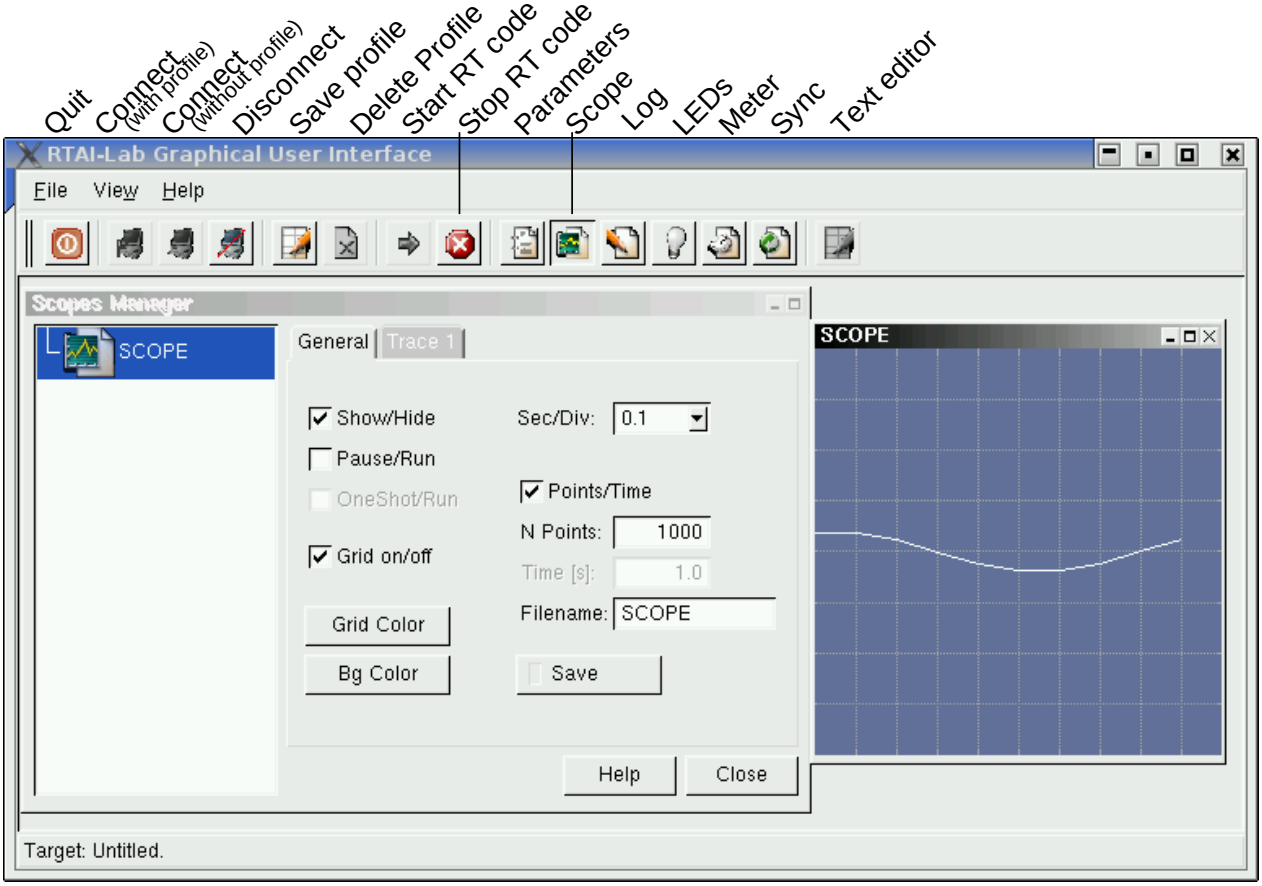


Figure 7: `xrtailab` with scope manager window and a scope displaying a 1 Hz sinewave. Most functions of the row of buttons are duplicated in the drop-down menus. Keyboard shortcuts are also available.

3.4.4 Change parameters

`xrtailab` lets you change "on the fly" the parameters that were defined in the Scicos block diagram. For example, the Sine block has 5 parameters: amplitude (Value[0]), frequency (Value[1]), phase (Value[2]), bias (Value[3]), and delay (Value[4]).
Click on the Parameters button. The Parameters Manager window opens. You might have to resize `xrtailab`'s window and reposition the Parameters Manager window to obtain a convenient layout similar to that shown in figure 8. Change Value[0] to 3 and press the Enter key. The sinewave's amplitude will increase accordingly. Change Value[1] to 2. This will double the sinewave's frequency to 2 Hz and you will notice that the wave lost its smoothness. This is an undersampling effect induced by the fact that the clock's period, as defined in the Scicos block diagram, is only of 0.1 s or 10 Hz.
Note that the Parameters Manager also lets you download block parameters from real-time tasks running on remote hosts and upload them back after modification.

3.4.5 Stop executable

Click on the Stop hexagonal icon to disconnect `xrtailab` from the real-time target and terminate it. Note that it is possible to only disconnect `xrtailab` from a target (alt-d) and leave it running. You can re-connect to it later. Finally, you may also interrupt the real-time executable directly at the terminal where you launched its execution: type ctrl-c.

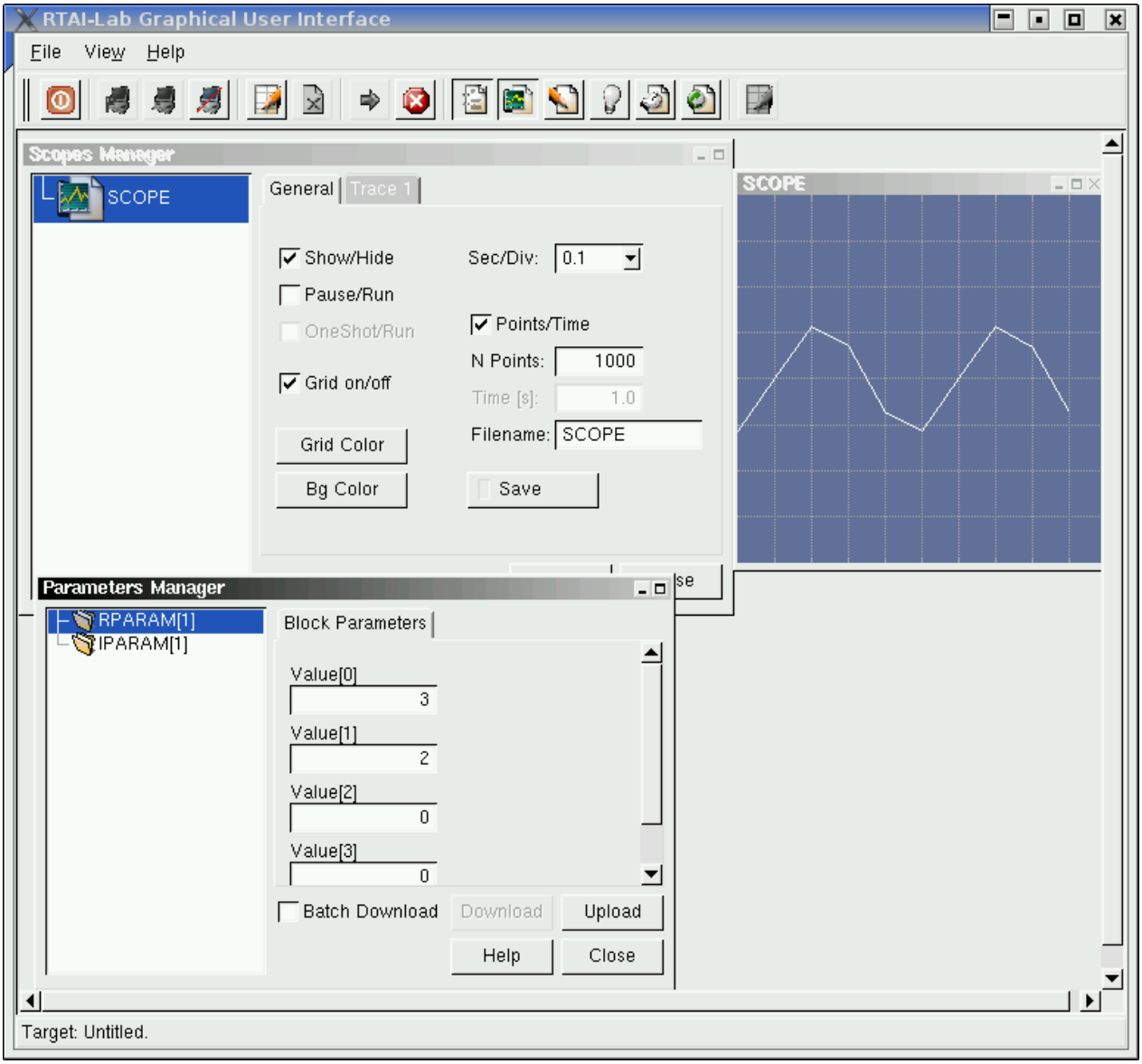


Figure 8: `xrtailab` with Parameters Manager window. Parameters that have been changed are sinewave amplitude (Value[0]) from 1 to 3 and frequency (Value[1]) from 1 to 2 Hz. The scope reveals an undersampled sinewave that results from the real-time executable's clock being set to only 10 Hz.

4 Examples

In this section we provide a few basic examples that show how to:

- Use FIFOs
- Use semaphores
- Represent a continuous-time plant as a discrete-time model

4.1 FIFOs

FIFOs provide a way of passing data to other tasks and programs. Design the diagram of figure 9a.

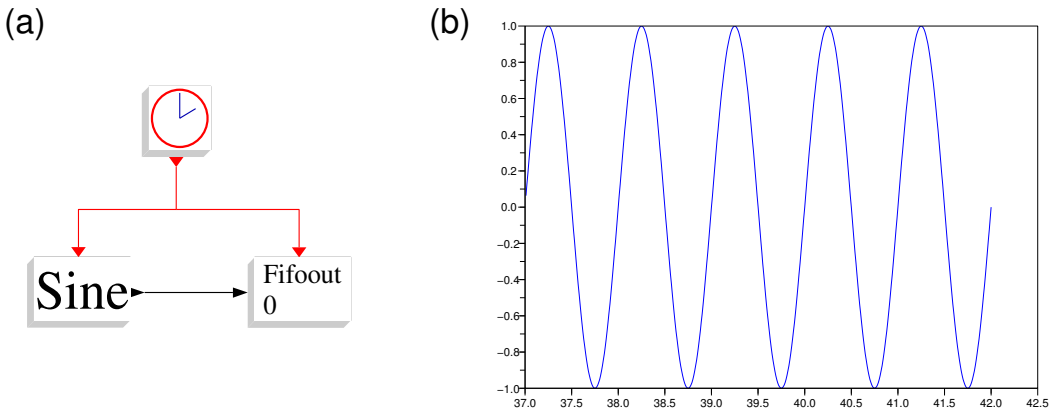


Figure 9: Preliminary diagram of a task that outputs sinewave data to a FIFO. An external program is then used to read the data from `/dev/rtf0`

- Click on Simulate → Eval to ensure all blocks are registered. You may change parameters in the Sine block.
- Set clock period to 0.01
- Frame a Superblock around Sine and FIFOout
- Open the Superblock and click menu Diagram → Rename. Replace `Untitled` with `fifo1`
- Close the Superblock's window
- Diagram → Save As: `fifo1.cos`
- Menu RTAI → RTAI CodeGen and click on the superblock to generate code and compile it to produce the executable `fifo1`

Now copy the program `readfifo.c` and compile it with: `gcc -o readfifo readfifo.c`

```
/* readfifo.c -- Read a FIFO and print its data */
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <signal.h>

static int end;
```

```
struct data{
     float t;
     float u[1];
};

static void do_end(int dummy) { end = 1; }

int main (void)
{
     int fifo;
     struct data val;

     if ((fifo = open("/dev/rtf0", O_RDONLY)) < 0) {
          fprintf(stderr, "Error opening /dev/rtf0\n");
          exit(1);
     }
     signal(SIGINT, do_end);
     while (!end) {
          read(fifo, &val, sizeof(val));
          printf("%f\t%f\n", val.t, val.u[0]);
     }
     return 0;
}
```

- In a terminal type: `./fifo1 -v`
- In a second terminal type: `./readfifo`
  Two columns of data should start printing. Interrupt the program by typing ctrl-c.
- In the second terminal now type: `./readfifo > fifo.dat`
  Interrupt the program after 3 to 10 seconds by typing ctrl-c.
- In Scicos, menu Diagram → Quit
- In Scicoslab's scilex window, type "n" and then type at Scicoslab's "-->" command prompt to produce the plot in Fig. 9b:

```
-->x = read('fifo.dat', -1, 2);
-->plot(x(:,1), x(:,2))
```

4.2 Semaphores

Semaphores are used as a signalling method between tasks. Design the diagram of figure 10.

- Open the SEM signal block and ensure that IP addr is 127.0.0.1
- Open the SEM wait block and ensure that IP addr is 0
- Open the left LED block and call it LED1
- Frame a Superblock on the left side around blocks Sine, SEM signal, and LED. Frame a Superblock on the right side around blocks SEM wait, LED, Sine, and Scope.
- Set each clock to a Period of 0.001
- Simulate → Eval

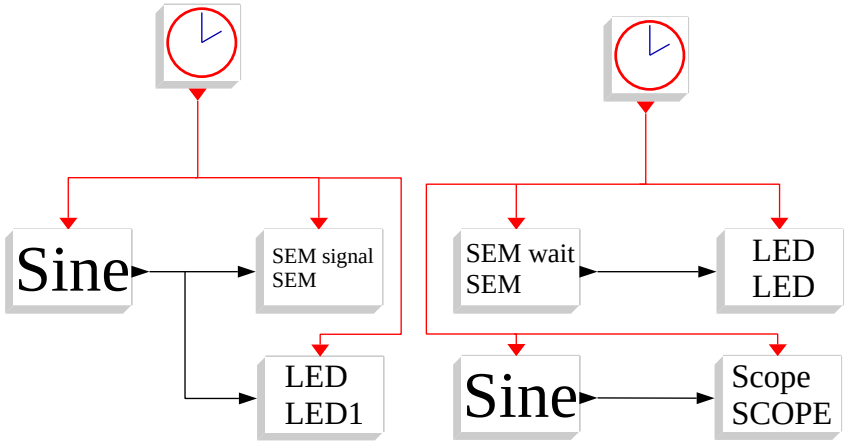


Figure 10: Preliminary diagram of two tasks that use semaphores. Left: the signalling task uses a sinewave to increment a semaphore. Right: the waiting task reads the semaphore and produces an output only when the sinewave in the signalling task is positive.

- Open the left Superblock and Diagram → Rename it to `semsig`. Note: you might have to Diagram → Replot to recenter the diagram. Warning: do not put a dash "-" in the name of a block, RTAI CodeGen has problems with that. Close the window.
- Put a name tag under the Superblock: Object → Identification, click on the left Superblock and set ID to `semsig`
- Open the right Superblock and rename it to semwait. Close the window.
- Save the diagram to sem.cos
- RTAI → RTAI CodeGen the left Superblock
- RTAI → RTAI CodeGen the right Superblock
- In a terminal type: `semsig -v -n IFTASL -d LED`
- In a second terminal type: `semwait -v -e`
- In a third terminal type: `xrtailab -v`
- In `xrtailab`:
  - Connect to `semwait`: type alt-c and set the "Led identifier" field to `LED`
  - Open and display an LED: click on the light bulb button then check Show/Hide. The *signalling task*'s LED should blink every second
  - Open and display a scope. You should see the *waiting task*'s truncated sinewave scrolling in a jerky manner every second (Fig.11).
  - Disconnect from `semwait`: type alt-d
  - Connect to `semsig`: type alt-c. In the Task Identifier field type: `IFTASL` and click on the OK button.
  - Open and display an LED. LED1 should blink at 1 Hz.
  - Open the "parameters manager": type alt-p. Increase Value[1] of RPARAM[1] from 1 to 2 and press Enter. By doing this you have increased semsig's Sine frequency from 1 to 2 Hz. LED1 should now blink at 2 Hz.
  - Disconnect from `semsig`: type alt-d
  - Connect again to `semwait`: type alt-c. Set Task Identifier to `IFTASK`
  - Open again an LED and a scope. The LED now blinks at 2 Hz and the sinewave has twice as many kinks.

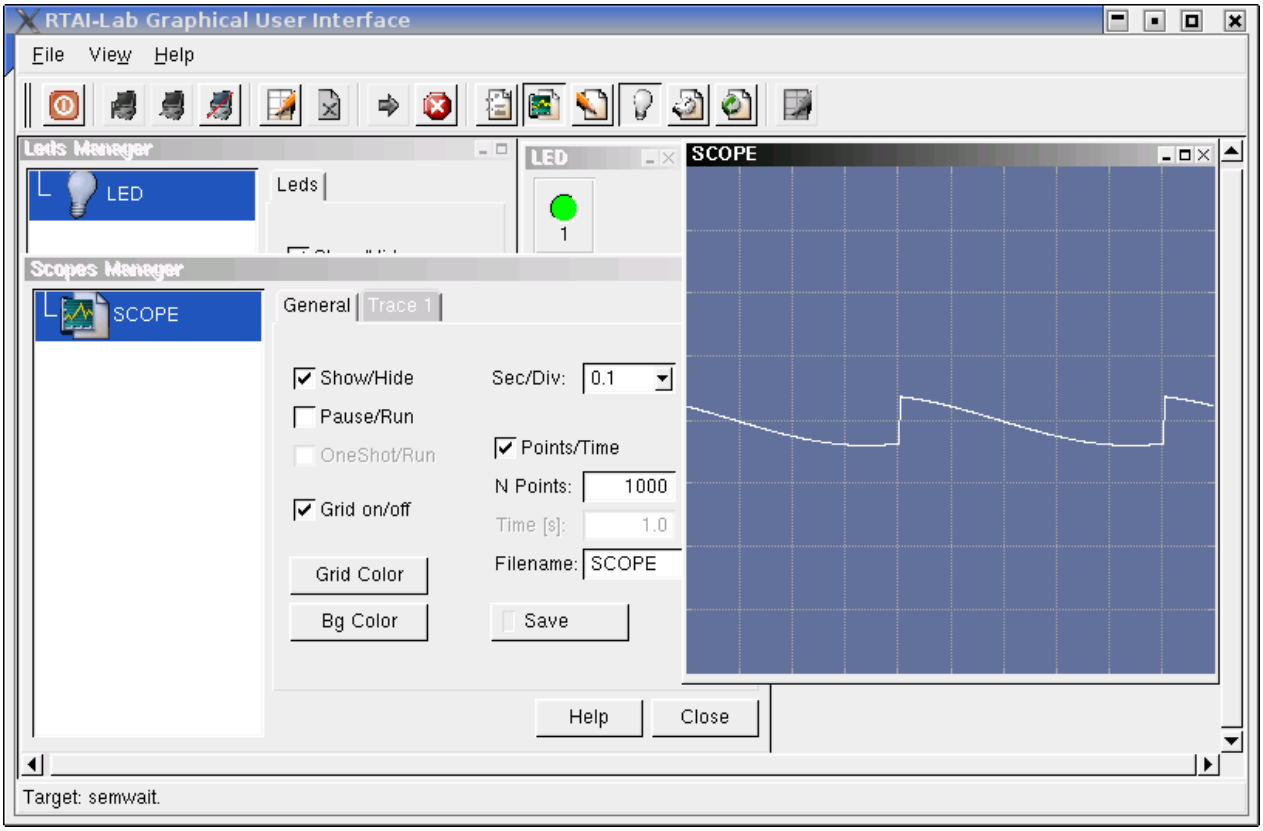


Figure 11: `xrtailab` scope display of the sinewave generated by the semwait Superblock (the waiting task). The green LED reflects the state of the LED block in the semsig Superblock (the signalling task). The waiting task operates only when the “SEM wait” semaphore is positive (light green LED). The result is a jerky sinewave synchronized to the semsig task’s semaphore.

4.3 From continuous- to discrete-time

4.3.1 Discretize the system

Say a continuous-time feedback system includes a plant modeled by the transfer function:

$$G(s) = \frac{20}{s^2 + 4s}.$$

Use Scicoslab to discretize the system with a sampling time of 1 ms.

```
-->s = poly(0,'s');
-->G = [20/(s^2 + 4*s)]
 G  =
      20
    ------
          2
    4s + s
-->Gc = syslin('c',G);
-->T = 0.001;
-->Gd = ss2tf(dscr(tf2ss(Gc),T))
 Gd  =
     0.0000100 + 0.0000100z
    ------------------------
                              2
    0.9960080 - 1.996008z + z
```

The system's discrete-time transfer function is therefore:

$$G(z) = 10^{-6} \frac{10z + 10}{z^2 - 1.996z + 0.996}.$$

4.3.2 Block diagram and simulation

Replicate the block diagram of figure 12 (you will use palettes Sources, Sinks, Linear, and Branching).

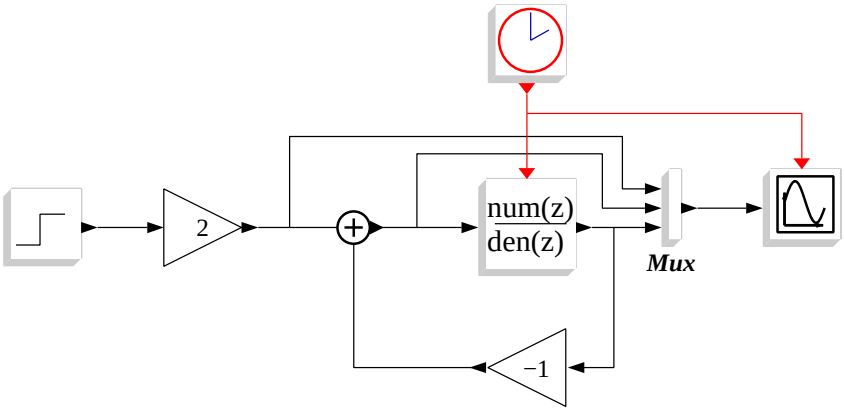


Figure 12: Scicos block diagram to simulate the step response of a discretized plant.

- **Clock block:** set Period = 0.001, Init time = 0
- **Step block:** set step time = 1, Initial value = 0, Final value = 1.
- **Gain block:** set gain to 2

- **Discrete SISO transfer function:** set numerator to: `0.00001 + 0.00001*z`
  set denominator to: `0.996008 - 1.996008*z + z^2`
- **Feedback gain block:** set gain to -1
- **Scope block:** set Ymin = -1, Ymax = 3, Refresh period = 5, Buffer size = 20
- **Menu Simulate → Setup:** set Final integration time = 5
- **Menu Simulate → Eval**

Use Scicos to visualize the system's response to a square signal: menu Simulate → Run produces the plot of figure 13.

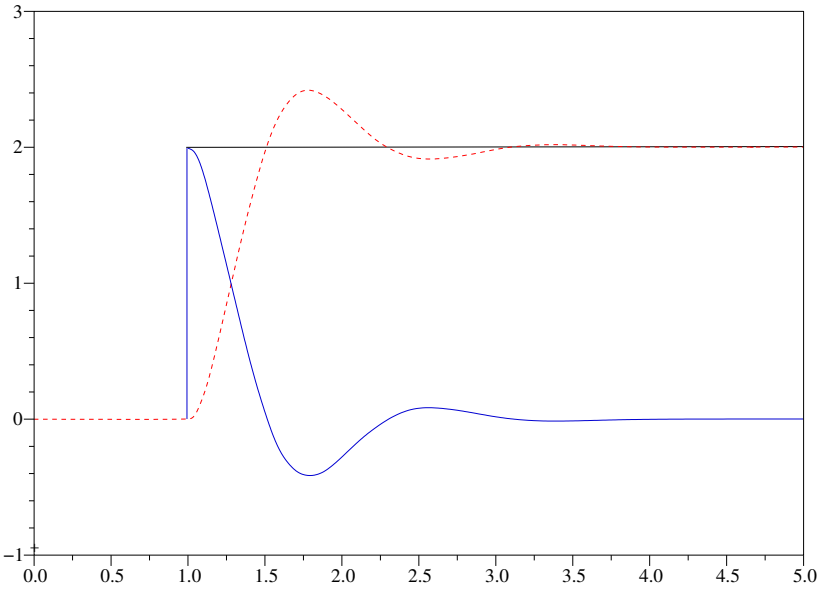


Figure 13: Step response (dashed red line) of block diagram shown in Fig. 12

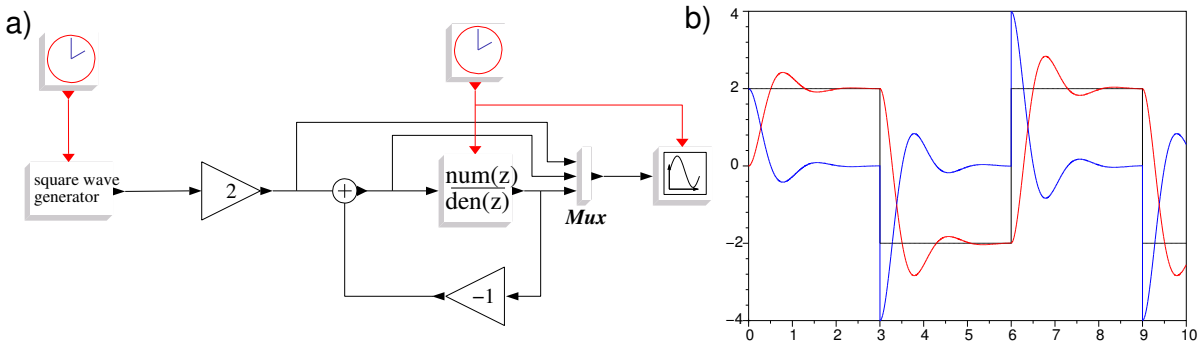


Figure 14: a) Block diagram of Fig. 12 modified to display the response to a square wave. b) Resulting plot showing square wave and step response (red line). Note: This figure was assembled by exporting the block diagram to Postscript and converting it with `pstoedit -f fig`. The plot was directly exported to `Xfig` format. Both figures were then merged using `Xfig` and the numbering on the axes was enlarged using `Xfig`'s "Update" button.

Modify the block diagram so as to simulate the response to a square wave input (Fig. 14a):

- **Left clock:** Period = 3, Init time = 0
- **Square wave generator:** Amplitude = 1
- **Scope:** Ymin = -4, Ymax = 4, Refresh period = 10, Buffer size = 20
- **Menu Simulate** → **Eval** generates Fig. 14b.

4.3.3 Real-time model and simulation

Modify again the block diagram using blocks from the RTAI-Lib palette (Fig. 14a):

- Replace the left clock and square wave generator with an RTAI-Lib Square block. Set Square block parameters: Amplitude = 1, Period = 6, Impulse width = 3, Bias = 0, Delay = 0
- Replace the multiplexer and scope with an RTAI-Lib Scope block. Set input ports = 3
- **Menu Diagram** → **Region to Super Block:** frame all blocks except the clock. You then obtain Fig. 14b
- Open Super Block. Menu Diagram → Rename and set diagram name to `rtex3`. Close Super Block window.
- **Menu RTAI** → **RTAI CodeGen** and compile
- In a terminal type: `./rtex3 -v`
- In another terminal type: `xrtailab`
- In xrtailab: connect to target (alt-c), open the scope manager and show the scope. In the Scopes Manager you can click on Trace tabs to adjust each trace's color (Fig. 15).

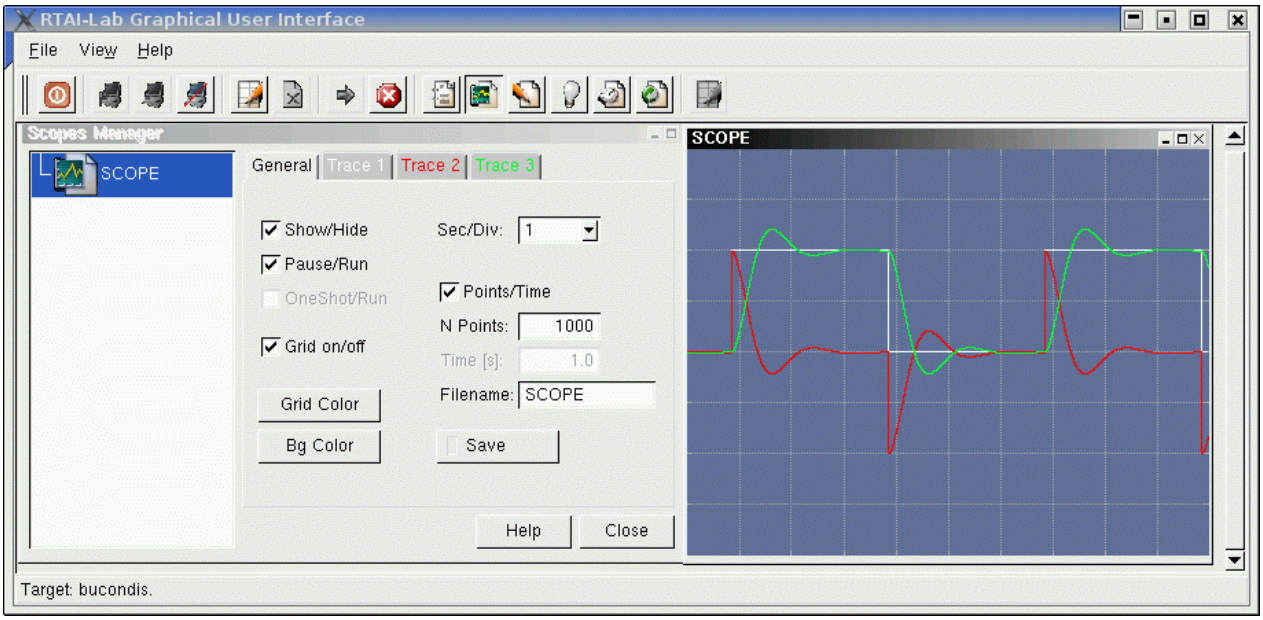


Figure 15: `xrtailab` Scopes Manager and scope displaying real-time plots of the block diagram shown in Fig. 14a

4.4 DC Motor control

In this example we show how to adapt a Simulink block diagram to Scicos and RTAI. This example is based on the Control Tutorials for Matlab® and Simulink®, hosted at Carnegie Mellon University and University of Michigan [Messner and Tilbury, 1998]: www.library.cmu.edu/ctms. Please refer to the "DC Motor position modeling in Simulink" example. Sections 7.2 and 10.2 of [Campbell et al., 2006] also provide in-depth information on using Scicos for control.

4.4.1 Motor position model

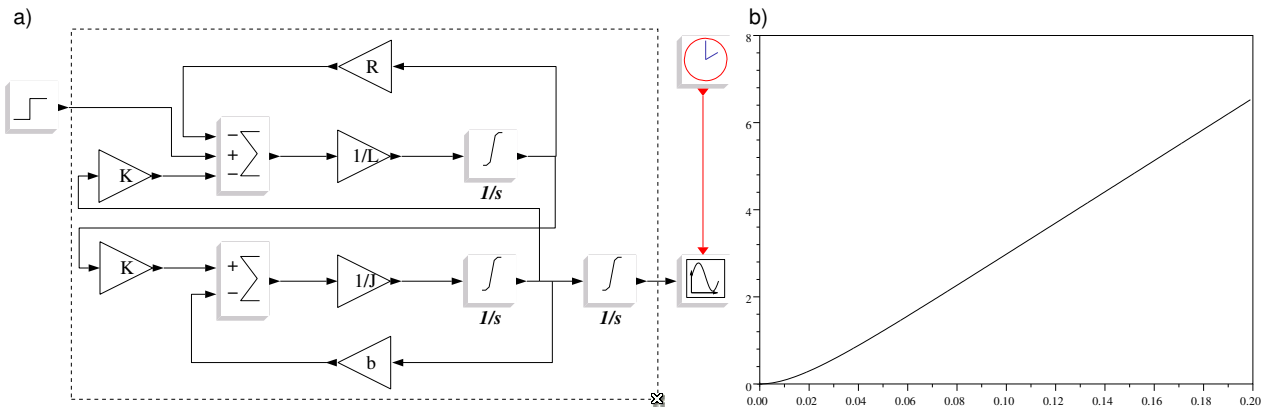


Figure 16: a) Continuous time model of DC motor position as described at www.library.cmu.edu/ctms. b) Open-loop step response of DC motor.

- Replicate the model of Fig. 16a
- Menu Edit → Context: enter values for constants:
  `J = 3.2284E-6; b = 3.5077E-6; K = 0.0274;`
  `R = 4; L = 2.75E-6`
- Menu Simulate → Setup: Final integration time = 0.2
- Step block: Step time = 0, Initial value = 0, Final value = 1
- Clock block: Period = 0.001, Init time = 0
- Scope block: Ymin = 0, Ymax = 8, Refresh period = 0.2
- Menu Simulate → Eval
- Menu Simulate → Run: produces the plot of Fig. 16b

4.4.2 Digital model

- Menu Diagram → Region to Super Block: draw a frame around the computational blocks as shown in Fig. 17a. You then obtain a superblock.
- Menu Diagram → Save As: save to dcmot-model.cos
- Menu Object → Get Info, then click on the superblock. A window opens, select "Others [Yes]" and click on OK (Fig.17a). A second window opens, remember the superblock's object number (here 43, Fig. 17b) and click on OK
- Menu Diagram → Quit to exit Scicos, then type "n" to get past the junk in the Scicoslab window.
- In Scicoslab type:
  (replace the "43" in the 3rd line with your superblock's object number):

a) b)

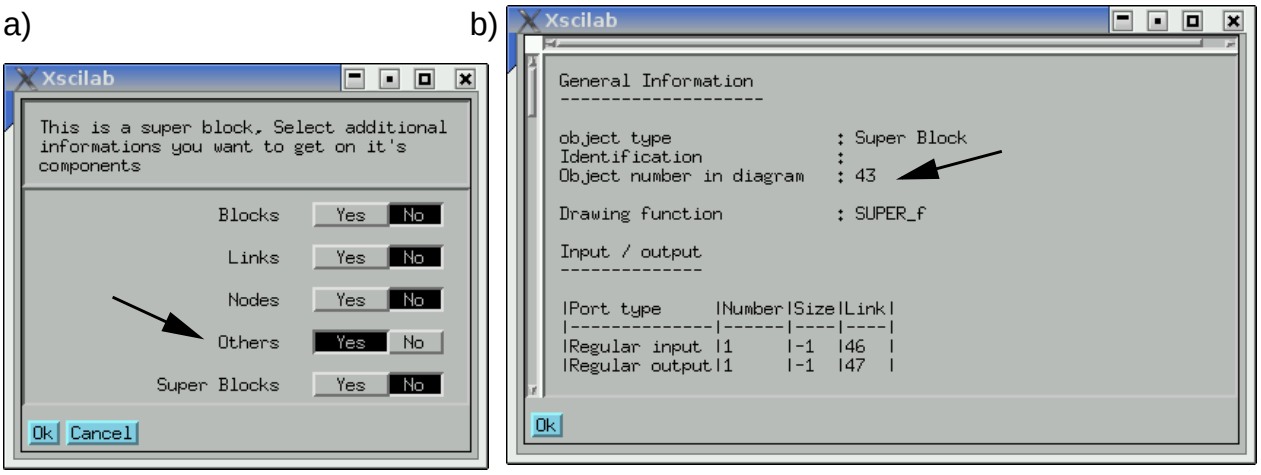


Figure 17: How to obtain information about a superblock prior to its discretization: menu Object → Get Info then a) select Others [Yes] and b) remember block's object number (43)

```
load dcmot-model.cos
J = 3.2284e-6; b = 3.5077e-6; k = 0.0274; R = 4; L = 2.75e-6
scs_m = scs_m.objs(43).model.rpar
sys = lincos(scs_m);
dtf = ss2tf(dscr(sys,0.001))
```

You then obtain the discrete-time transfer function from voltage input to motor position output:

```
dtf   =
                                      2
 9.454E-10 + 0.0010214z + 0.0010389z
 -----------------------------------
                             2    3
     0.9424937z - 1.9424937z + z
```

which after simplification and reordering corresponds to:

$$\frac{0.001z + 0.001}{z^2 - 1.9425z + 0.9425}.$$

You can recover the transfer function's plot of Fig. 16b with:

```
td = 0:T:0.2
u = ones(1, size(td,2));
plot2d([td'],[(dsimul(tf2ss(dtf),u))']);
```

4.4.3 Digital control

Build upon the superblock resulting from the block diagram of Fig. 16a and generate the diagram of Fig. 18a:

- Left clock: Period = 0.1, Init time = 0
- Square wave generator: Amplitude = 1
- Discrete transfer function num(z)/den(z):
  Numerator(z) = `450*(z-0.85)*(z-0.85)`
  Denominator(z) = `(z+0.98)*(z-0.7)`

- Scope block: Ymin = -1.5, Ymax = 1.5, Refresh period = 0.2
- Menu Simulate → Setup: Final integration time = 0.2
- Menu Simulate → Eval
- Menu Simulate → Run: produces the animated plot of Fig. 18b

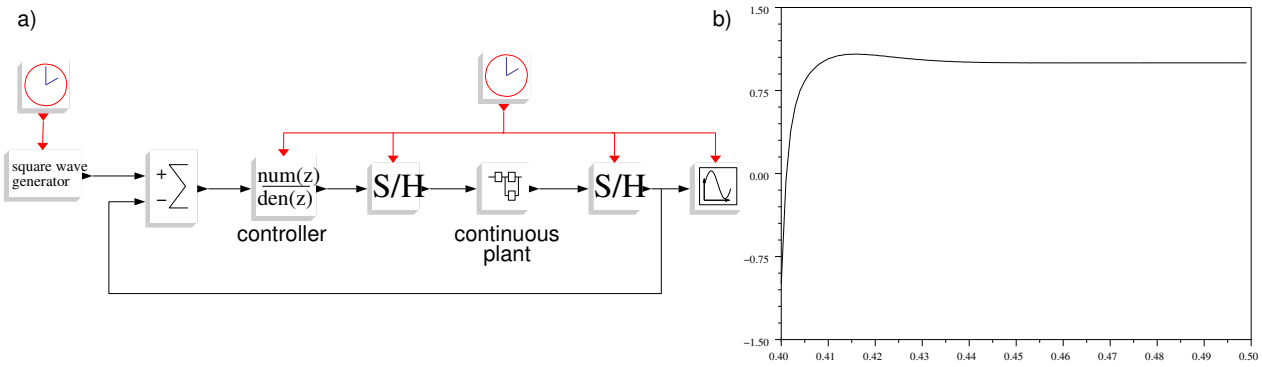


Figure 18: a) Closed-loop system with digital controller. b) Step response

4.4.4 RTAI-Lab controller

In its current version, RTAI-Lab requires the block diagram to be simplified (Fig.19).

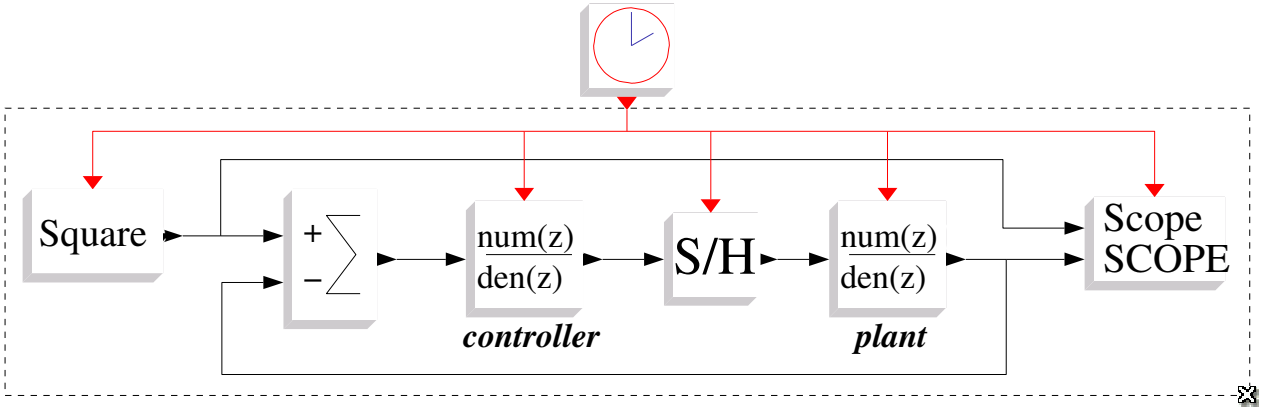


Figure 19: Simplified DC motor control loop.

You will have to:

- Clock Period = 0.001
- Replace Scicos's square wave generator with RTAI-Lab's real-time Square block
- Unfactorize the controller's transfer function with:
  Numerator(z) = `450*z^2 - 765*z + 325.125`
  Denominator(z) = `z^2 + 0.28*z - 0.686`
- Replace the plant's continuous model by a discretized model:
  Numerator(z) = `0.001*z + 0.001`
  Denominator(z) = `z^2 - 1.9425*z + 0.9425`
- Replace Scicos's scope by RTAI-Lab's real-time scope: input ports = 2
- Menu Diagram → Region to Super Block: frame all blocks except the clock

- Open superblock and Menu Diagram → Rename: dcmot
- Menu RTAI → RTAI CodeGen
- In a terminal type: `dcmot -v`
- In another terminal type: `xrtailab`
- In xrtailab: connect to target (alt-c), open the scope manager and show the scope. In the Scopes Manager you can click on Trace tabs to adjust each trace's color (Fig. 20).
- Disconnect and stop the real-time task by clicking on the red button.

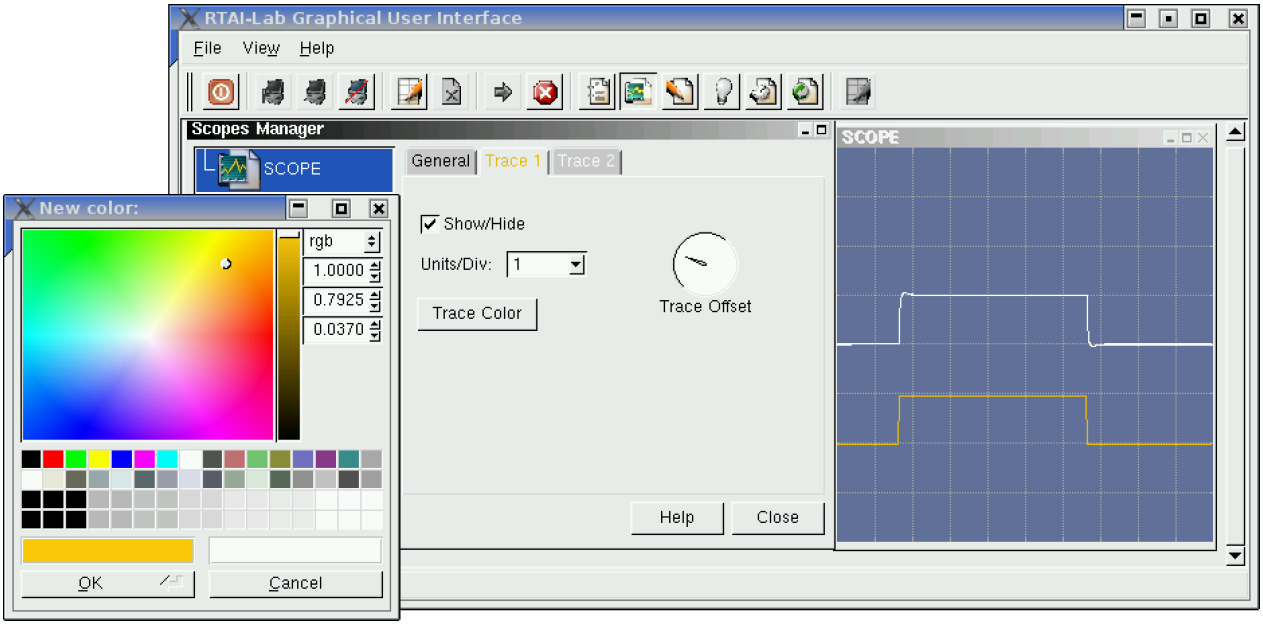


Figure 20: `xrtailab` lets you adjust trace offset and color as well as units/division. Here the square wave input is set to orange. The output signal (white) features a slight overshoot.

- Finally, return to the block diagram, open the superblock, add a Comedi D/A block under the Scope, and connect it similarly (Fig. 21a).
- Close the window and RTAI CodeGen the superblock.
- Connect the analog output and analog ground of your signal acquisition and generation hardware to a real oscilloscope. For example with the National Instruments PCMCIA DAQcard 6024E, connect pins 22 (DAC0OUT) and 55 (AOGND).
- In one terminal (Fig. 21c): `xrtailab -v`
- In another terminal (Fig. 21d): `dcmot -v`
- You should now see something that looks like a square wave on your oscilloscope. Settings: vertical: 500mV/div, horizontal: 250ms/div.
- In `xrtailab`: connect, open the scope, adjust visualization parameters in a similar manner to the oscilloscope. You can grab the scope's window corner to enlarge the display.

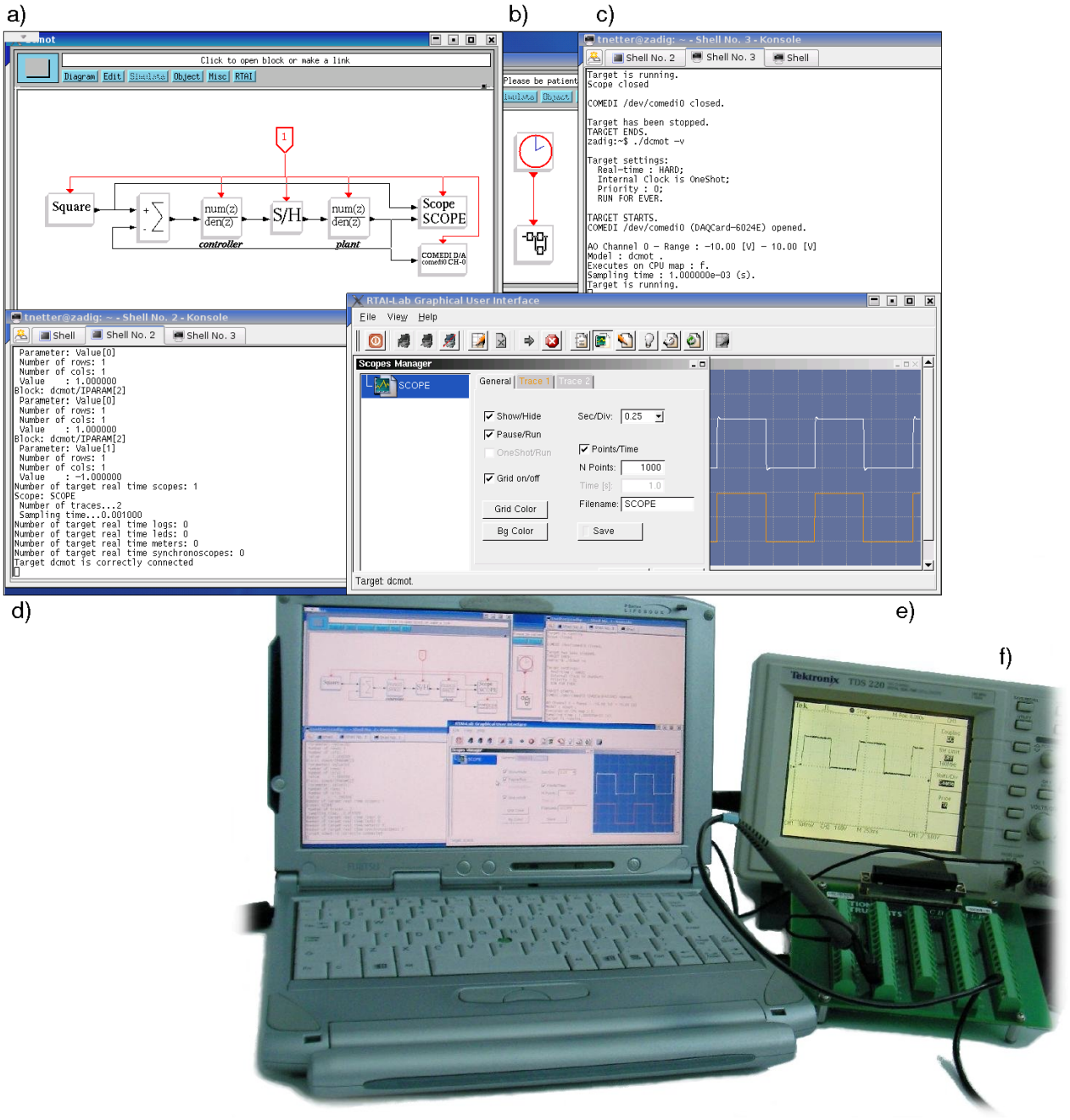


Figure 21: Signal generation and visualization on an oscilloscope. a) Add a Comedi A/D block and connect it. Close the window. b) RTAI CodeGen on the Super Block. c) Execute the real-time task: `dcmot -v`. d) `xrtailab -v`. e) `xrtailab`: adjust units/div and sec/div. f) Visualize the signal on an oscilloscope (ensure all grounds are well connected). The laptop is a Fujitsu P-2040 featuring a Transmeta Crusoe TM-5800 CPU. Data acquisition hardware: National Instruments PCMCIA DAQCard-6024E.

5 Custom blocks

Custom Scicos blocks can be programmed, added to palettes, and used to generate real-time code.
In section 5.2 we present an example where part of the code is generated from templates. In section 5.2.2 we show how to create blocks from scratch, and add them to the Scicoslab + RTAI-Lab installation. Finally, we explain how palettes are created.
Note that Chapter 9.5 of [Campbell et al., 2006] provides much information on Scicos blocks.
See www.scicos.org

5.1 Overview of Scicos blocks

A Scicos block is defined by two functions:

- A Scicos *interfacing function*. It is coded in Scicoslab language and compiled. A block's code specifies:
  - The block's graphical appearance
  - The number of data inputs and outputs
  - The number of event inputs and outputs
  - Dialog windows to adjust appearance, inputs and outputs, and computation parameters
- A *computational function*. It is coded in a separate C file (it used to be enclosed with in a dedicated `getCode` function after the Scicos interfacing function). It is this code that is compiled and executed for simulations or even used for "Hardware in the loop (HIL)" tests (see www.scicos.org. This code may also call external C libraries.

  The C code is segmented according to a flag's value, thereby defining the job type such as [Campbell et al., 2006; Nikoukhah and Steer, 1998]:
  - Initialization (flag = 4)
  - Output update (flag = 1)
  - State update (flag = 2)
  - Integrator calls (flag = 0)
  - Mode and zero-crossing (flag = 9)
  - Event scheduler (flag = 3)
  - Ending (flag = 5)

Source code for Scicos and RTAI-Lab blocks may provide useful guidance and is available in the `.sci` files in directories: `SCILAB_DIR/macros/scicos_blocks` and `SCILAB_DIR/macros/RTAI`

5.2 Hardware example: parallel port LED blinker

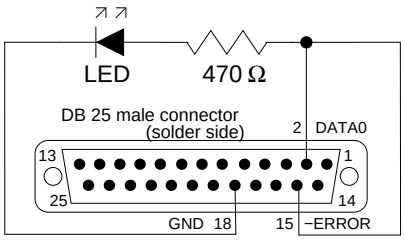
This example uses the parallel port to output and input a bit that will blink an LED.
The example shows how to:

- Build the circuit
- Program Scicos blocks
- Produce C code to access the parallel port

5.2.1 Parallel port + LED circuit

Build a circuit with:

- a DB-25 male connector
- an LED
- a 470 Ω resistor



Check your BIOS settings for parallel port mode and address. The safest choice is SPP (Standard Parallel Port) and base address 0x378 (other valid addresses are 0x278 or 0x3BC). This example outputs to pin 2/DATA0 (base address, bit 0) and inputs from pin 15/-ERROR (base+1, Bit 3). Parallel port pinout addresses are detailed in Appendix B if you wish to extend this example.

5.2.2 Create the environment for the new blocks

First, you can create a folder structure where you can put all the new files. In this example we create a new folder called “ScicosPP” which contains two subfolder “devices” and “scicosdev” under “develop” in my home directory.

5.2.3 Generate the interface files

After moving in the subfolder “scicosdev” we can create the two interfaces files for our new blocks.

rtai_pport_out.sci

```
function [x,y,typ] = rtai_pport_out(job,arg1,arg2)
  x=[];y=[];typ=[];
  select job
  case 'plot' then
    exprs=arg1.graphics.exprs;
    pinout=exprs(1)
    paddr=exprs(2)
    standard_draw(arg1)
  case 'getinputs' then
    [x,y,typ]=standard_inputs(arg1)
  case 'getoutputs' then
    [x,y,typ]=standard_outputs(arg1)
  case 'getorigin' then
    [x,y]=standard_origin(arg1)
  case 'set' then
    x=arg1
    model=arg1.model;graphics=arg1.graphics;
    exprs=graphics.exprs;
    while %t do
      [ok,pinout,paddr,exprs]=..
      getvalue('Set block parameters',..
      ['Parallel port PIN [0..7]:';
       'Parallel port Address'],..
      list('vec',-1,'str',1),exprs)
      if ~ok then break,end
      if exists('inport') then in=ones(inport,1), else in=1, end
      out=[]
      [model,graphics,ok]=check_io(model,graphics,in,out,1,[])

      if ok then
        graphics.exprs=exprs;
        model.rpar=[];
        model.ipar=[pinout;hex2dec(strsubst(paddr,'0x',''))];
        model.dstate=[];
        x.graphics=graphics;x.model=model
        break
      end
    end
  case 'define' then
    pinout=1
    paddr='0x378'
    model=scicos_model()
    model.sim=list('rt_pport_out',4)
    if exists('inport') then model.in=ones(inport,1), else model.in=1, end
    model.out=[]
    model.evtin=1
    model.rpar=[]
    model.ipar=[pinout;hex2dec(strsubst(paddr,'0x',''))]
    model.dstate=[];
    model.blocktype='d'
    model.dep_ut=[%t %f]
    exprs=[sci2exp(pinout),paddr]
    gr_i=['xstringb(orig(1),orig(2),[''Pport_out'' ; string(pinout) ; paddr],sz(1),sz(2),''fill'');']
    x=standard_define([3 2],model,exprs,gr_i)
  end
endfunction
```

rtai_pport_in.sci

```
function [x,y,typ] = rtai_pport_in(job,arg1,arg2)
  x=[];y=[];typ=[];
  select job
  case 'plot' then
    exprs=arg1.graphics.exprs;
    pinin=exprs(1)
    paddr=exprs(2)
    standard_draw(arg1)
  case 'getinputs' then
    [x,y,typ]=standard_inputs(arg1)
  case 'getoutputs' then
    [x,y,typ]=standard_outputs(arg1)
  case 'getorigin' then
    [x,y]=standard_origin(arg1)
  case 'set' then
    x=arg1
    model=arg1.model;graphics=arg1.graphics;
    exprs=graphics.exprs;
    while %t do
      [ok,pinin,paddr,exprs]=..
      getvalue('Set block parameters',..
      ['Parallel port Input [1...5]:';
       'Parallel port Address'],..
      list('vec',-1,'str',1),exprs)
      if ~ok then break,end
      in=[],
      if exists('outport') then out=ones(outport,1), else out=1, end
      [model,graphics,ok]=check_io(model,graphics,in,out,1,[])

      if ok then
        graphics.exprs=exprs;
        model.rpar=[];
        model.ipar=[pinin;hex2dec(strsubst(paddr,'0x',''))];
        model.dstate=[];
        x.graphics=graphics;x.model=model
        break
      end
    end
  case 'define' then
    pinin=1
    paddr='0x378'
    model=scicos_model()
    model.sim=list('rt_pport_in',4)
    model.in=[],
    if exists('outport') then model.out=ones(outport,1), else model.out=1, end
    model.evtin=1
    model.rpar=[]
    model.ipar=[pinin;hex2dec(strsubst(paddr,'0x',''))]
    model.dstate=[];
    model.blocktype='d'
    model.dep_ut=[%t %f]
    exprs=[sci2exp(pinin),paddr]
    gr_i=['xstringb(orig(1),orig(2),[''Pport_in'' ; string(pinin) ; paddr],sz(1),sz(2),''fill'');']
    x=standard_define([3 2],model,exprs,gr_i)
  end
endfunction
```

5.2.4 Compile the interface files

First you must create the following “buildmacros.sce” file.

```
// Scilab ( http://www.scilab.org/ ) - This file is part of Scilab
// Copyright (C) 2007-2008 - INRIA - Allan CORNET <allan.cornet@inria.fr>
//
// This file must be used under the terms of the CeCILL.
// This source file is licensed as described in the file COPYING, which
// you should have received as part of this distribution.  The terms
// are also available at
// http://www.cecill.info/licences/Licence_CeCILL_V2-en.txt

if (isdef('genlib') == %f) then
  exec(SCI+'/modules/functions/scripts/buildmacros/loadgenlib.sce');
end
//-----------------------------------
toolbox_dir = pwd();
genlib('pportdev',toolbox_dir,%f,%t);
//-----------------------------------
```

where “pport” is the name of your new interface library.
The following “Makefile” can be used to compile the files:

```
install:
        sl -nw -f buildmacros.sce

clean:
        rm -f *.bin lib names
```

5.2.5 Generate the implementation files

RTAI-Lab provides the `/usr/realtime/bin/gen_dev` script to generate a C skeleton for new RTAI-Lab devices
Move to the folder “devices” and insert the following two files:

pport_out.c

```
/*
COPYRIGHT (C) 2007  Roberto Bucher (roberto.bucher@supsi.ch)

This library is free software; you can redistribute it and/or
modify it under the terms of the GNU Lesser General Public
License as published by the Free Software Foundation; either
version 2 of the License, or (at your option) any later version.

This library is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public
License along with this library; if not, write to the Free Software
Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA.
*/

#include <machine.h>
#include <scicos_block4.h>
#include <sys/io.h>

static void init(scicos_block *block)
{
  block->ipar[0]=block->ipar[0];
}

static void inout(scicos_block *block)
{
  double * u = block->inptr[0];
  unsigned char val;

  val = inb(block->ipar[1]);

  if (u[0] > 0) val |= (0x01 << block->ipar[0]);
  else          val &= ~(0x01 << block->ipar[0]);
  outb(val,block->ipar[1]);
}

static void end(scicos_block *block)
{
  outb(0x00,block->ipar[1]);
}

void rt_pport_out(scicos_block *block,int flag)
{
  if (flag==1){          /* set output */
    inout(block);
  }
  else if (flag==5){     /* termination */
    end(block);
  }
  else if (flag ==4){    /* initialisation */
    init(block);
  }
}
```

pport_in.c

```
/*
COPYRIGHT (C) 2007  Roberto Bucher (roberto.bucher@supsi.ch)

This library is free software; you can redistribute it and/or
modify it under the terms of the GNU Lesser General Public
License as published by the Free Software Foundation; either
version 2 of the License, or (at your option) any later version.

This library is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public
License along with this library; if not, write to the Free Software
Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA.
*/

#include <machine.h>
#include <scicos_block4.h>
#include <sys/io.h>

static void init(scicos_block *block)
{
}

static void inout(scicos_block *block)
{
  double * y = block->outptr[0];
  unsigned char val;

  val = inb(block->ipar[1]+1);
  val ^= 0x80;
  val = (val >> (block->ipar[0]));
  if(val & 0x08) y[0] = 1.0;
  else           y[0] = 0.0;
}

static void end(scicos_block *block)
{
}

void rt_pport_in(scicos_block *block,int flag)
{
  if (flag==1){          /* set output */
    inout(block);
  }
  else if (flag==5){     /* termination */
    end(block);
  }
  else if (flag ==4){    /* initialisation */
    init(block);
  }
}
```

5.2.6 Compile the implementation files

You must create a new C-library with the implememention code of your new blocks.
The following “Makefile” can be used to perform this task:

```
LIB = libmysci.a
all: $(LIB)

RTAIDIR = $(shell rtai-config --prefix)

SRC = $(wildcard *.c)

OBJ = $(SRC:.c=.o)
```

```
DBG = -g

CC_FLAGS = -c $(DBG) -I. -I$(RTAIDIR)/include -O2

%.o: %.c
        cc $(CC_FLAGS) $<

$(LIB): $(OBJ)
        ar -r $(LIB) $(OBJ)

install: $(LIB)
        sudo cp $(LIB) $(RTAIDIR)/lib

clean:
        rm -f $(LIB) $(OBJ)
```

where “libmysci” is the name of your new library.
Now simply launch “make” and “make install” to generate the library and copy it into the RTAI folder (usually under /usr/realtime/lib).

5.2.7 Integrate the new files in the Scicoslab environment

In order to be able to use the new blocks in the Scicoslab environment, you must add the following line in the “.scilab” file:

```
load('/home/bucher/develop/ScicosPP/scicosdev/lib')
```

5.2.8 The template Makefile for the new files

By compiling the Scicoslab block diagram containing the new blocks, you need to inform the linker about your new library. You need consequently a new template Makefile which substitute the provided rtai.mak and a new target file which substitute rtai.gen.
Your new target (saved for example as “pport.gen“) file must be

```
pport.mak
standalone.cmd
```

and the corresponding template Makefile looks like:

```
# Makefile generate from template rtai.mak
# ========================================

all: ../$$MODEL$$

RTAIDIR = $(shell rtai-config --prefix)
C_FLAGS = $(shell rtai-config --lxrt-cflags)
SCIDIR = $$SCILAB_DIR$$

RM = rm -f
FILES_TO_CLEAN = *.o ../$$MODEL$$

CC = gcc
CC_OPTIONS = -O -DNDEBUG -Dlinux -DNARROWPROTO -D_GNU_SOURCE

MODEL = $$MODEL$$
OBJSSTAN = rtmain44.o common.o $$MODEL$$.o $$OBJ$$

SCILIBS = $(SCIDIR)/libs/scicos.a $(SCIDIR)/libs/poly.a $(SCIDIR)/libs/calelm.a
```

```
$(SCIDIR)/libs/blas.a $(SCIDIR)/libs/lapack.a $(SCIDIR)/libs/os_specific.a
OTHERLIBS =
ULIBRARY = $(RTAIDIR)/lib/libsciblk.a $(RTAIDIR)/lib/liblxrt.a
$(RTAIDIR)/lib/libmysci.a

CFLAGS = $(CC_OPTIONS) -O2 -I$(SCIDIR)/routines -I$(SCIDIR)/routines/scicos
$(C_FLAGS) -DMODEL=$(MODEL) -DMODELN=$(MODEL).c

rtmain44.c: $(RTAIDIR)/share/rtai/scicos/rtmain44.c $(MODEL).c
        cp $< .

../$$MODEL$$: $(OBJSSTAN) $(ULIBRARY)
        gcc -static -o $@  $(OBJSSTAN) $(SCILIBS) $(ULIBRARY) -lpthread -lm
        @echo "### Created executable: $(MODEL) ###"

clean::
        @$(RM) $(FILES_TO_CLEAN)
```

where the only difference to the original ”rtai.mak“ file is in the line with the ”ULIBRARY“ where we added the name of the new library ”libmysci.a“.

5.2.9 Create a scicos block palette for the new files

Create a new diagram with Diagram → New. Load new blocks into this diagram with Edit → Add New Block. You may also slide blocks from pre-existing palettes into this new diagram. Then save your diagram with Diagram → Save as Palette (for example as ”pport.cosf“).
The next step is the generation of the ”palette tree“ which can be performed using the following script:

```
unix('ls -1 *.sci > x.x');
txt=mgetl('x.x');
create_gif(txt,'./gif_icons')
mdelete x.x
```

The last step is to integrate the new palettes into the scicoslab environment. Simply put the following lines in the ”.scilab“ file

```
scicos_pal($+1,:)=['SUPSI-Lib','/home/bucher/ScicosPP/develop/scicosdev/pport.cosf']
%scicos_gif($+1)=['/home/bucher/develop/ScicosPP/scicosdev/gif_icons/' ];
```

5.3 Scicos C block data structure

When a block is linked to another in the Scicos GUI, the link allows data to be passed from one block to its successor. When a block diagram is converted to C code, the block's data-passing scheme along with some parameters are represented in C by the `block` data structure of type `scicos_block` (see `SCI/routines/scicos/scicos_block.h`). See usage examples in the `.c` files in `SCI/routines/scicos` and in the RTAI-Lab `.sci` files in `SCI/macros/RTAI`. [Campbell et al., 2006] gives more details on how to use this structure.
Members of the `block` structure are listed below:

```
  int block->nevprt; /* binary coding of activation inputs, */
   /* -1 if internally activated  */
  int block->nz; /* size of the discrete time state */
  double* block->z; /* vector of size nz: discrete-time state */
  int block->nx; /* size of the continuous-time state */
  double* block->x; /* vector of size nx: continuous-time */
  double* block->xd; /* vector of size nx: derivative of */
/* continuous-time state */
```

```
  double* block->res; /* vector of size nx: only used for */
/* internally implicit blocks */
  int block->nin; /* number of inputs */
  int *block->insz; /* input sizes */
  double **block->inptr; /* table of pointers to inputs */
  int block->nout; /* number of outputs */
  int *block->outsz; /* output sizes */
  double **block->outptr; /* table of pointers to outputs */
  int block->nevout; /* number of activation output ports */
  int block->nrpar; /* number of real parameters */
  double *block->rpar; /* real parameters of size nrpar */
  int block->nipar; /* number of integer parameters */
  int *block->ipar; /* integer parameters of size nipar */
  int block->ng; /* number of zero-crossing surfaces */
  double *block->g; /* zero-crossing surfaces */
  int *block->jroot;
  char block->label[41];
```

A Notes on Linux configuration and installation

This appendix details how to configure a new RTAI kernel that you will boot as an alternative to your standard Linux kernel. In other words, you should not use RTAI as the only kernel on you system. You should use your computer's boot menu (Lilo or Grub) to select either your current Linux kernel, RTAI-Linux kernel, or other operating system.

A.1 Kernel configuration

As a first try, you should configure a minimum RTAI kernel. You may then add hardware drivers (CD, sound, etc.) after some testing. For help on configuring a kernel see:

- www.linuxdocs.org/HOWTOs/Kernel-HOWTO.html
- www.linuxheadquarters.com/howto/tuning/kernelconfig.shtml

Note that a convenient starting point if you have already compiled a Linux kernel of a similar 2.x generation is to copy the `.config` to the new linux-2.x.xx-rtai directory and type `make oldconfig`.
The following guidelines are for a 2.6.x kernel. Now launch the configuration tool (`make xconfig` or `make menuconfig`).

- `cd /usr/src/linux-2.x.xx-rtai`
- `make xconfig` or `make menuconfig`
- **General setup:** select "Prompt for development..."
- **General setup:** set "Local version" to `-rtai`
- **Enable loadable module support:** select "Enable module support", "Module unloading", and "Automatic module loading". Deselect "Module versioning support"; RTAI modules are not version dependent.
- **Processor type and features:** Select your Subarchitecture Type (PC-Compatible) and Processor family. Select "Preemption Model (Preemptible kernel (Low-Latency Desktop))". You might need "High Memory Support (4GB)" if you use a PCMCIA data acquisition card. Deselect "Use register arguments (EXPERIMENTAL)". Possibly deselect "Local APIC support on uniprocessors".
- **Power Management options:** Select ACPI Support and features relevant to your hardware. Leave APM deselected. Leave CPU Frequency scaling unselected. Warning: ACPI support may be a problem on laptops that use the "screen closed" button to put the computer into sleep or standby modes.
- **Bus options:** Leave the default. Check the support for your hardware. Laptops need PCCARD (PCMCIA/CardBus) support and PC-card bridges. e.g. CardBus yenta-compatible bridge support
- **Device Drivers:**
  - **Generic driver options:** keep default
  - **Memory Technology Devices (MTD):** not needed
  - **Parallel port support:** unselect Parallel port support. The standard parallel port is a useful device for realtime debugging and experimenting. You must leave it unselected so that Comedi's drivers can directly access the port.
  - **Plug and Play support:** keep default
  - **Block devices:** select your devices. Fedora Core III needs Ram Disk Support and Initial RAM Disk (initrd) to boot.
  - **ATA/ATAPI/MFM/RLL Support:** select the main item "ATA/ATAPI/MFM/RLL support" and all items relevant to your system.
  - **SCSI device support:** select "SCSI device support" and keep the default selections according to your computer's SCSI devices.
  - **Multi-device support (RAID and LVM):** only needed in special cases. For Fedora Core III default installation you need this option enabled with LVM (Device Mapper) enabled too.

  - **Network device support:** keep defaults. Explore the devices submenu until you find your network interface. The command `lspci` may reveal the name of your ethernet controller.
  - **Amateur Radio, Irda, Bluetooth, ISDN subsystem, and Telephony support:** Leave disabled. You may enable these later.
  - **Input device support:** Ensure that Mouse is selected.
  - **Character devices:** Ensure "/dev/agpart AGP Support" and "Direct Rendering Manager (XFree 4.1.0 and higher DRI support)" are selected. See coments below about video card selection.
  - **I2C support:** keep unselected; there are reports of difficulties when used with RTAI
  - **Multimedia devices:** keep unselected. You may enable these later
  - **Graphics support:** keep unselected, possibly enabling later. With this option you can use the advanced features of your video card, but sometimes this creates compatibility problems.
  - **Sound:** preferably enable as module and select suitable drivers.
  - **USB Support:** preferably enable as module.
- **File Systems:**
  - **Second extended fs support:** select
  - **Ext3 journalling file system support:** select it and "Ext3 extended attributes"
  - **Reiserfs support:** the Suse distribution uses it, maybe your Linux distribution doesn't need it
  - **CD/ROM-DVD Filesystems:** select "ISO 9660..." and sub-items
  - **DOS/FAT/NT Filesystems:** select as needed

  Keep other default selections

If you recompile the kernel to add some features, you need also to recompile, install and check both RTAI and Comedi.

A.2 Configuring the GRUB boot manager

This section gives an example to configure the GRUB boot manager's configuration file usually located in `/etc/grub.conf` (Fedora Core II distribution) or /etc/grub/grub.conf (Debian distribution).
See also www.gnu.org/software/grub

```
# grub.conf
# Note that you do not have to rerun grub after making changes to this file
# NOTICE:  You have a /boot partition.  This means that
#          all kernel and initrd paths are relative to /boot/, e.g.:
#          root (hd0,1)
#          kernel /vmlinuz-version ro root=/dev/hda3
#          initrd /initrd-version.img
#          boot=/dev/hda
default=2
timeout=10
splashimage=(hd0,1)/grub/splash.xpm.gz
# The original F.C. II Generic-Modular Non-Realtime Kernel
title Fedora Core (2.6.5-1.358)
        root (hd0,1)
        kernel /vmlinuz-2.6.5-1.358 ro root=LABEL=/ rhgb quiet
        initrd /initrd-2.6.5-1.358.img
# RTAI patched Kernel
title Linux (2.6.23.14-RTAI-3.3)
        root (hd0,1)
        kernel /vmlinux-2.6.23.14-rtai
# VERY useful memory tester program
title Memtest86
        root (hd0,1)
```

```
        kernel /memtest86+-1.11
#
title Windows XP-Home
        rootnoverify (hd0,0)
        chainloader +1
```

A.3 *udev* and RTAI

udev is a dynamic device detection scheme (see www.kernel.org/pub/linux/utils/kernel/hotplug/udev.html) that is used by several Linux distributions and works with the 2.6 Linux kernel.
udev automatically creates device inodes in the `/dev` directory upon device detection. It later deletes those inodes as the devices are unused.
RTAI uses inodes in `/dev` for FIFOs, message passing between user and kernel space, and to interact with data acquisition hardware. It is therefore necessary to create persistent inodes. You can for example launch the script below prior to loading RTAI and Comedi modules (make it executable with `chmod 777 rtai-inode`):

```
# rtai-inode: RTAI inode creation for UDEV systems, creates /dev/rtf(n)

rm -f /dev/comedi* /dev/rtf* /dev/rtai_shm

for n in ‘seq 0 9‘; \
do \
        rm -f /dev/rtf$n; \
mknod -m 666 /dev/rtf$n c 150 $$n; \
done ; \

# create shared memory inode
mknod -m 666 /dev/rtai_shm c 10 254

# create Comedi inodes
for i in ‘seq 0 15‘;
do \
        rm -f /dev/comedi$i; \
        mknod -m 666 /dev/comedi$i c 98 $i ; \
done;
```

Basically, *udev* has an init script that reinstalls all devices in `/dev`. A user can create devices in `/lib/udev/devices` and the init script will copy them into `/dev` at boot time. The above script can be modified to create devices under `/lib/udev/devices`.

B Reference: standard parallel port

DB-25 Pin allocation.

Pin	Name	Func.	Register	Hardware Inverted
1	nStrobe	In/Out	Control	Yes
2	Data 0	Out	Data	
3	Data 1	Out	Data	
4	Data 2	Out	Data	
5	Data 3	Out	Data	
6	Data 4	Out	Data	
7	Data 5	Out	Data	
8	Data 6	Out	Data	
9	Data 7	Out	Data	
10	nAck	In	Status	
11	Busy	In	Status	Yes
12	P_Out/_End	In	Status	
13	Select	In	Status	
14	nAuto	In/Out	Control	Yes
15	Error	In	Status	
16	nInit	In/Out	Control	
17	nSelect	In/Out	Control	Yes
18-25	Ground	Gnd		

```
In/Out: TTL level input / Open Collector output
Out   : TTL level output
In    : TTL level input

Hardware Inverter: the in/out voltage level is inverted respect to
                   normal logic.
```

I/O BASE valid address: 0x378 - 0x278 - 0x3BC

BASE+0 : Data Register

Bit	Pin	Name	Fun.	Register
0	2	Data 0	Out	Data
1	3	Data 1	Out	Data
2	4	Data 2	Out	Data
3	5	Data 3	Out	Data
4	6	Data 4	Out	Data
5	7	Data 5	Out	Data
6	8	Data 6	Out	Data
7	9	Data 7	Out	Data

BASE+1 : Status Register

Bit	Pin	Name	Fun.	Register	Hardware Inverted
0		Reserved			
1		Reserved			
2		Reserved			
3	15	Error	In	Status	
4	13	Select	In	Status	
5	12	P-Out/-End	In	Status	
6	10	nAck	In	Status	
7	1	Busy	In	Status	Yes

BASE+2 : Control

Bit	Pin	Name	Fun	Register	Hardware Inverted
0	1	nStrobe	In/Out	Control	Yes

1	14	nAuto	In/Out	Control	Yes	5	En_DATA_READ
2	16	nInit	In/Out	Control		6	unused
3	17	nSelect	In/Out	Control	Yes	7	unuded
4		En_IRQ(nACK)					

References

Stephen L. Campbell, Jean-Philippe Chancelier, and Ramine Nikoukhah. *Modeling and Simulation in Scilab/Scicos*. Springer, Berlin, Germany, 2006. URL `www.scicos.org`.

Gene F. Franklin, J. David Powell, and Michael L. Workman. *Digital control of dynamic systems*. Addison Wesley Longman, Reading, MA, 1998.

Bill Messner and Dawn Tilbury. *Control tutorials for MATLAB and Simulink*, 1998. URL `www.library.cmu.edu/ctms`.

Ramine Nikoukhah and Serge Steer. *SCICOS - A Dynamic System Builder and Simulator, User's Guide*, 1998. URL `scilabsoft.inria.fr/doc/scicos/scicos.html`. Also as: scilabsoft.inria.fr/doc/scicos/scicos.pdf.

Giovanni Racciu and Paolo Mantegazza. *RTAI 3.3 User Manual*, 2006. URL `www.rtai.org`. in Documentation → Reference Documents.

G. Sallet. *Ordinary differential equations with Scilab*. Université de Saint-Louis / INRIA Lorraine, Université de Metz, 2004. URL `www.math.univ-metz.fr/~sallet/ODE_Scilab.pdf`.

Pasi Sarolahti. Real-time application interface. Technical report, University of Helsinki, Dept. of Comp. Sci., 2001. URL `www.cs.helsinki.fi/u/sarolaht/papers/rtai.pdf`.

BibTex entry for this document:

```
@Manual{RTAILab:2006,
  title =   {RTAI-Lab tutorial: Scicoslab, Comedi, and real-time control},
  author =  {Roberto Bucher and Simone Mannori and Thomas Netter},
  year =  2006,
  url =  {www.rtai.org/RTAILAB}
}
```

C About this document / Bug reports

This tutorial and associated source code can be downloaded in PDF and HTML from
https://www.rtai.org/RTAILAB
This tutorial should be considered "work in progress" and subject to frequent updates, we therefore request that you do not make it available on other web sites.
This document is copyright ©2005 - 2008 Roberto Bucher, Simone Mannori, Thomas Netter.

The first edition (June 14, 2006) of this document was written and tested with: Linux 2.6.12.6 patched with RTAI 3.3, Scilab 4.0, Comedi CVS downloaded 2006-06-09.
This 3rd edition (Mar. 14, May 23, 2007, Feb. 2008) was updated to RTAI-3.5 and partly tested with: Linux 2.6.23.14 patched with RTAI 3.6, Scilab/Scicos 4.1.2, Comedi CVS, gcc-4.1.3, Mesa 7.0.2, efltk 2.0.7.

C.1 About the authors

Roberto Bucher <roberto.bucher at supsi.ch> is a lecturer and researcher in automatic control with the Department of Innovative Technology at the University of Applied Sciences of Southern Switzerland (SUPSI), in Manno, near Lugano. He is the main developer of RTAI-Lab.
Homepage: www.dti.supsi.ch/~bucher
Simone Mannori <smannori at f2n.it> is a research engineer with the METALAU project at the French National Institute for Research in Computer Science and Control (INRIA), in Rocquencourt, near Paris. He is currently developing Scicoslab's "hardware-in-the-loop" soft real-time functionalities.
Thomas Netter <tnetter at ifi.unizh.ch> is a researcher with the Artificial Intelligence Laboratory, University of Zurich, Switzerland. His work focuses on real-time control, vision, and unmanned air vehicles (UAV). Homepage: www.ifi.unizh.ch/ailab/people/tnetter

C.2 Bug reports, suggestions

RTAI-Lab queries: If you notice software bugs, have questions, wish to suggest improvements to RTAI-Lab such as submit new blocks for the RTAI-Lib palette, contact the main developer:
<roberto.bucher at supsi.ch>.
For bug reports please provide Linux kernel version, RTAI version, RTAI patch number, CPU type, data acqusition hardware type, Scicoslab version, gcc/g++/cpp versions, the block diagram that may cause the bug (.cos file), outputs using verbose option "-v", and possibly kernel logs resulting from `tail -f /var/log/syslog`.
RTAI-Lab tutorial queries: If you notice typos, errors, omissions, or have contributions to make to this tutorial, contact the author of this tutorial: Thomas Netter <tnetter at ifi.unizh.ch>
RTAI queries: Check www.rtai.org and its associated mailing-list
Scicoslab queries: Check www.scicoslab.org, www.scicos.org, and the newsgroup
comp.soft-sys.math.scilab

C.3 Software licenses

RTAI-Lab, RTAI, and xrtailab are subject to the GNU Lesser General Public License.
Comedi is subject to the GNU General Public License.
Scicoslab and Scicos are subject to a specific license. See scilabsoft.inria.fr/legal/license.html

C.4 Acknowledgements

The authors wish to thank:

- **Paolo Mantegazza**, RTAI project leader at the Department of Aerospace Engineering of the Politecnico di Milano, for contributing so much to the development of RTAI-Lab
- **Ramine Nikoukhah**, Scicos project leader with METALAU project at INRIA, for helping adapt Scicoslab to generate RTAI code and supporting RTAI-Lab's documentation effort
- **The Swiss National Science Foundation**, Thomas Netter is partly supported by a grant of the FNS/SNF.

D Useful files and links

- **RTAI-Lab** www.rtai.org/RTAILAB and `www.dti.supsi.ch/~bucher`
  - Pre-installation source of RTAI-Lab Scicos blocks: `/usr/src/rtai/rtai-lab/scilab/macros/RTAI`
  - Makefile to compile Scicos RTAI-Lab blocks: `/usr/src/rtai/rtai-lab/scilab/macros/RTAI/Makefile`
  - Post-installation location of RTAI-Lab Scicos blocks: `/usr/local/scilab-4.0/macros/RTAI`
  - RTAI-Lib Scicos blocks contain C code that make calls to C functions located in: `/usr/src/rtai/rtai-lab/scilab/devices/`.
    C prototypes are in `devices.h`. Associated C library is `libsciblk.a`, installed in `/usr/realtime/lib/`.
  - Install on Ubuntu + Matlab (by Arno Stienen): www.rtai.org/RTAILAB/RTAI-UbuntuGutsy-Matlab.txt
  - Use Windows + Matlab/Simulink/RTW for design and RTAI/RTAI-Lab as a target (by Giampiero Campa):
    www.rtai.org/RTAILAB/RTAI-TARGET-HOWTO.txt
- **Scicoslab** www.scicoslab.org / www.scicos.org
  - Tutorials:
    www.wolffdata.se/scilab.htm

    www.scicos.org/TUTORIAL/tutorial.html
  - French publications: www.saphir-control.fr/articles
  - Scicoslab start-up file that includes a pointer to RTAI-Lib palette: `$HOME/.Scilab/scilab-4.4b7/.scilab`
  - Scicoslab installation: `/usr/lib/scicoslab-gtk-4.4`
  - Code of standard Scicos blocks: `/usr/lib/scicoslab-gtk-4.4/macros/scicos_blocks/`
- **Comedi** www.comedi.org
  - Code of acquisition card drivers: `/usr/local/src/comedi/comedi/drivers`
- **RTAI** www.rtai.org
  - Draft of RTAI User Manual www.rtai.org → Documentation → Reference Documents
  - www.rtai.dk (somewhat outdated but useful stuff)
  - Hannes “Captain” Mayer’s notes: www.captain.at/rtai.php
  - Real-Time Linux tutorial (based on RTAI): www.isd.mel.nist.gov/projects/rtlinux
  - Links to several articles describing RTAI projects: pramode.net
  - RTAI internals:
    `www.aero.polimi.it/~rtai/documentation/articles/paolo-dissecting.html`
  - RTAI history: www.rtai.org → Documentation → Articles → RTAI History
  - Documentation: `/usr/realtime/share/doc/rtai-3.3`
  - Loadable modules are in: `/usr/realtime/modules`
- **Automatic control**
  - PI control with RTAI: linuxgazette.net/118/sreejith.html
  - Control tutorials for Matlab® and Simulink®: www.library.cmu.edu/ctms
    Hint: It would be great if someone wrote a Scicoslab version
  - Matlab files associated to the book Digital control of dynamic systems [Franklin et al., 1998]: www.mathworks.com/support/books/book1464.html
    Hint: It would be fantastic if someone translated the book’s m-files to Scicoslab
- **Miscellaneous links**

- RTAI-XML project to monitor RTAI tasks from various OS's: artist.dsi.unifi.it/rtaixml
- Moodss graphical monitoring application: moodss.sourceforge.net
  Note: it would be interesting to adapt Moodss to work with RTAI
- Matplotlib python 2D plotting library: matplotlib.sourceforge.net
- Qt interface to Matlab/Scicoslab/Octave: chainlink.sf.net